さうんど おんりぃ２　最終報告書

PM　株式会社インテム　菊地徹也
環境情報学部４年　橋山牧人
環境情報学部４年　藤原育実

目次

第1章. プロジェクトの概要

本章では本プロジェクトにおけるプロジェクトの定義，及びプロジェクトの計画について説明する．

1.1. 背景

「映像を用いないゲーム」は，5.1ch のサラウンド音響環境を利用することで，音源の発音する定位や位置情報，音色，リズムなどを聞きわけ，仮想の空間を体感するという新しいギミックを有するゲームである．映像のある一般的なゲームをプレイするよりも想像力を高めることができるなど，シリアスゲームの一環としての利用が想定されている． また，視覚にハンディキャップを有する方々にもゲームの面白さを伝えることができ，ゲームを楽しめるターゲットを増やすことができる．

1.2. 目的

- ユーザが繰り返し遊んでくれるような面白いと感じられるゲームを開発するその開発を通じて，プロジェクト進行，ゲーム製作を勉強していく．
- 人に使ってもらえるもの，人に楽しんでもらえるものを開発する．

1.3. 目標

ゲーム開発のプロセスやプロジェクトの進行を学習し，それによって，ソフトウェア開発手法との違いを学習する．それをメンバーごとに最終報告書にまとめる．

1.4. 体制

全体的な体制については以下の図を参照のこと．

- プロジェクトメンバー
  菊地　徹也（ＰＭ）
  橋山　牧人
  藤原　育実
- クライアント
  南雲　玲生（株式会社ユードー）

- ユーザ

  健常者と視覚にハンディキャップを有する方々

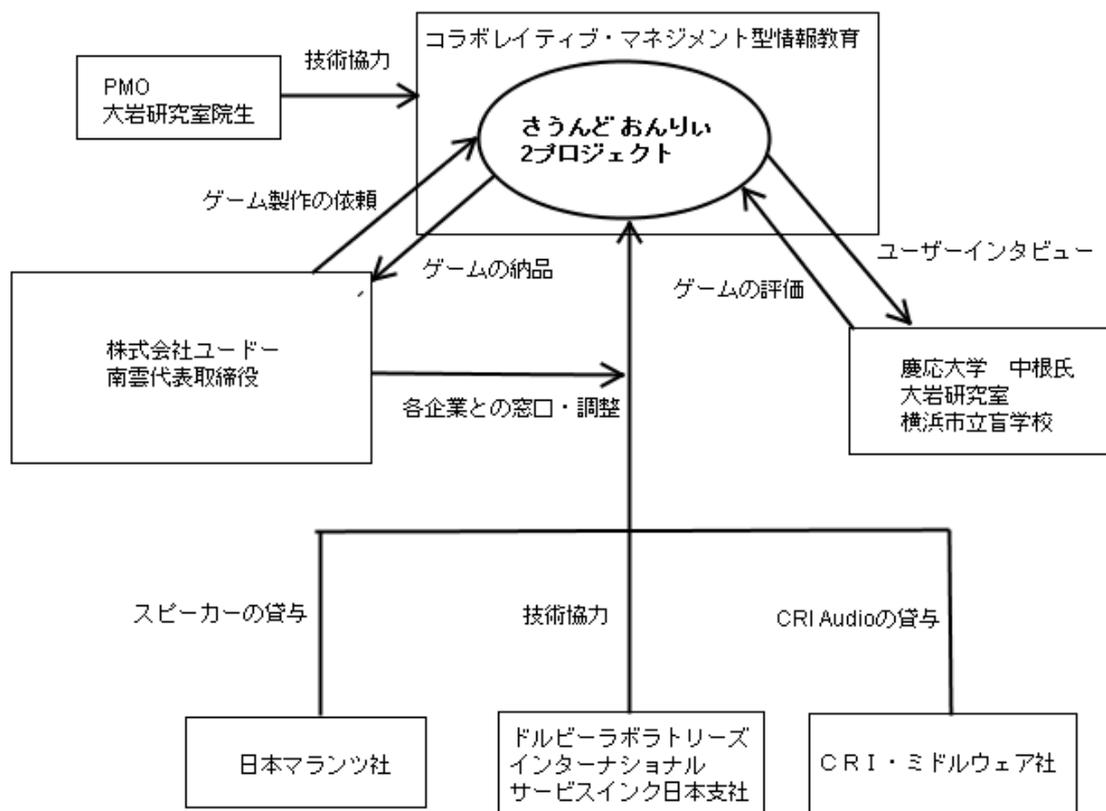



図 1-1　さうんど おんりぃ２プロジェクト　体制図

1.5. 開発環境

- コーディング
  - VisualStudio2003.NET（C++）
- ライブラリ各種
  - CRI Audio（5.1ch サラウンドライブラリ）
  - SDL（Simple Directmedia Layer）
  - Boost
- ヘッドフォン
  - 5.1ch サラウンドヘッドフォン
- データ共有
  - Wiki
  - SVN

1.6. 開発期間

開発期間は，2006 年 10 月 5 日から 2007 年 1 月 31 日である．

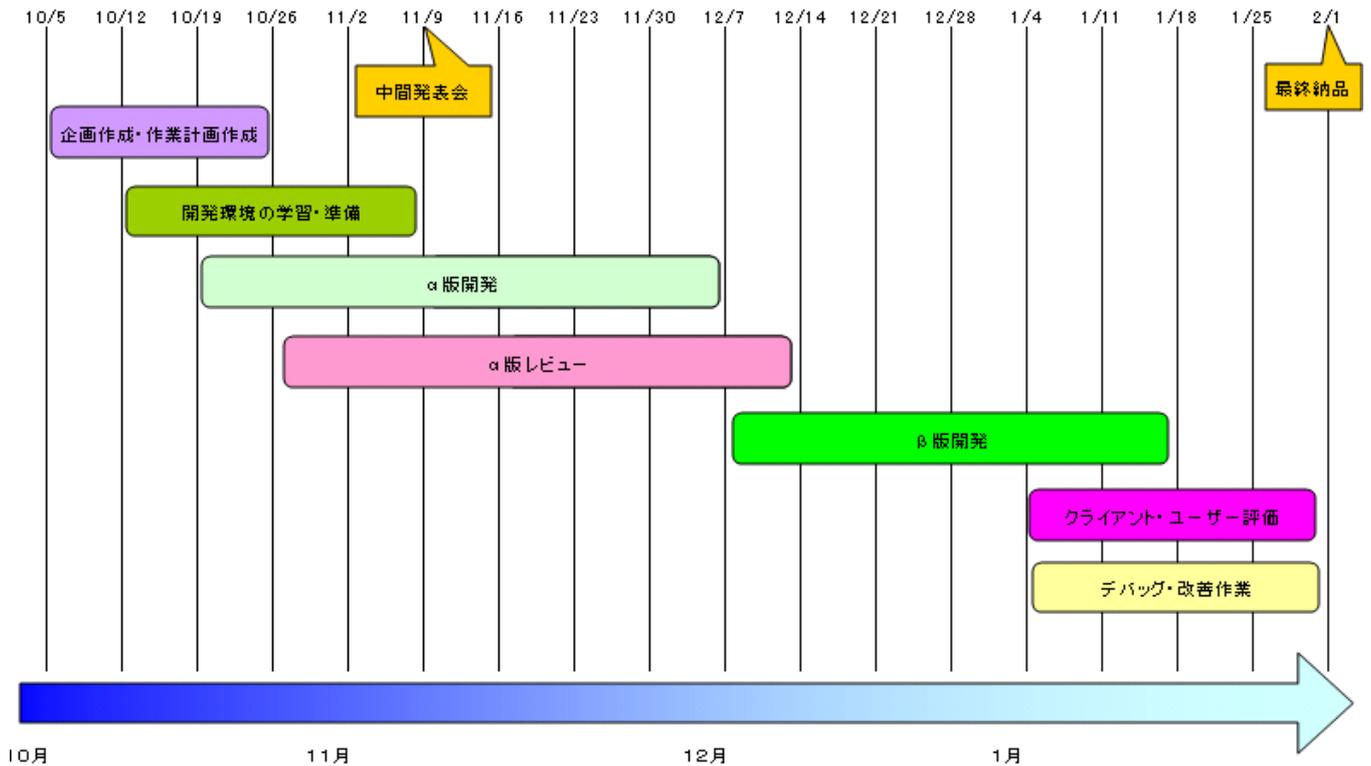



図 1-2 全体スケジュール（予定）

1.7. 成果物

- **ソフトウェア (映像を用いないゲーム)**
- **ゲーム企画書**
- **ソースコード**
- **リソースデータ (音楽, 効果音, イメージ)**
- **取扱説明書**
- **プロジェクト定義書**

1.8. プロジェクトの評価

1.8.1. 評価基準

- ユーザにゲームをプレイしてもらい，アンケートを記入してもらう
- アンケートの評価が去年の評価を上回ることを目標とする
  - 面白いと思う人が全体の 7 割以上 (前回は 5 割)
  - もう一度遊びたいと思う人が全体の 8 割以上 (前回は 7 割)
- 健常者，視覚にハンディキャップを有する人が共に面白いと感じるものを作る
  - 両者の評価に明確な差がなく，両者から目標とする評価を得る

1.8.2. 評価の実施について

- α版の評価を 11 月中旬に行う (慶応大学中根様，大岩研究室)
  - 意見を聞く
- β版の評価を 12 月下旬～1 月初旬 (横浜市立盲学校，大岩研究室)
  - アンケートについて，先学期作成したものを改良して利用する

1.9. プロジェクト運用のルール

- 毎週木曜日の進捗発表会の発表者はプロジェクトメンバーが持ち回りで担当しその準備も行う．
- メンバーは，毎週月曜5限目にミーティングを行う．
- プロジェクトメンバーのコミュニケーションには，さうんどおんりぃ2のメーリングリスト，Wiki を利用し，どれでも連絡とれないようであれば，携帯電話で連絡を取り合う．
- ファイルの共有は，Wiki，WebDAV，SVN などを利用し共有する．

第2章. プロジェクト実績報告

2.1. スケジュール

2.1.1. α版開発

当初計画した通りに進めることができなかった.

たくさんのゲームを中間報告会までに作成することを目的として開始したが，メンバーの作業時間が思うように取れないことが多く，実際には，ゲームとして，中途半端なものを中間報告会で発表することになってしまった.

中間報告会の反省を受けてα版の開発期間を延ばし，中間報告会までの反省と学習した知識で，新たな企画を作成し，現在の企画「虫捕りゲーム」の企画をまとめた.

2.1.2. β版開発

作業開始は，遅れてしまったが，α版開発時からの技術向上もあり，順調に進むことができた.

開発も後半に差し掛かると，自分達で，ゲームが面白くなるにはどうしたらいいのかと考え，さまざまな調節や工夫を取り入れ面白くするにはどうするかを考え作業をするようになった.

2.2. プロジェクト実績

このプロジェクトは，見積もり行っておりませんので実績だけ記載します.

2.2.1. プログラムステップ数（α版開発）

α版開発にあたって，いきなり大きな規模のゲームを作るのではなく，小規模なゲームを複数作ることにした. α版の期間で開発したのは，下記のゲームとなる.

- 方向当てゲーム
- 聖徳太子ゲーム

- サウンドシュート
- 聖徳太子ゲーム改

はじめはすぐ開発ができるということで java を用いて開発を行ってもらった．その後，C++での開発となり規模がだんだんと大きくなっていった. 以下はα版で開発したゲームのステップ数となる.

方向当てゲーム

ファイル形式	行数	コメント	空行	Logic	ファイル数
java	289	52	43	194	3
合計	289	52	43	194	3

聖徳太子ゲーム

ファイル形式	行数	コメント	空行	Logic	ファイル数
java	338	70	29	239	3
合計	338	70	29	239	3

さうんどシュート

ファイル形式	行数	コメント	空行	Logic	ファイル数
cpp	967	194	151	622	9
h	355	12	63	280	10
合計	1,322	206	214	902	19

聖徳太子ゲーム改

ファイル形式	行数	コメント	空行	Logic	ファイル数
cpp	1,025	199	154	672	9
h	360	12	59	289	10
合計	1,385	211	213	961	19

2.2.2. プログラムステップ数（β版開発）

α版の開発のあと，β版の開発となった．α版での反省，改善点などを踏まえ企画を考えひとつのゲームの完成を目指した．以下はβ版で開発したゲームのステップとなる．

β版開発ゲーム

ファイル形式	行数	コメント	空行	Logic	ファイル数
cpp	3,359	802	458	2,099	31
h	1,235	197	224	814	33
合計	4,594	999	682	2,913	64

2.3. 作業工数

2.3.1. α版作業工数

α版開発における作業工数は下記の表の通りである．作業的にはゲームの開発工数というよりは，どちらかというと勉強会のほうに時間を割かれている．実際，ミニゲームを複数開発する予定ではあったが，作業時間がとれず，ミニゲームを量産することができなかった．

α版作業工数表

作業項目	実績	
	橋山	藤原
ミーティング	12.0	10.5
ユーザレビュー	2.0	2.5
環境整備･学習	7.0	8
開発･企画	14.0	12.5
勉強会	28.5	14.5
その他･資料作成	2.5	4
合計	66.0	52.0

2.3.2. β版作業工数

実際にスタートする企画が決まり，作業を開始した．α版に比べると，作業時間も増え，プロジェクトの活動が活発になったように感じられる．

β版作業工数

作業項目	実績	
	橋山	藤原
ミーティング	13.5	13
ユーザレビュー	5	5
勉強会	11	-
環境準備	7.5	7
開発･企画	150.5	74
その他	47	39
合計	234.5	138

2.4. ユーザ評価

「映像のないゲーム」の成功評価をユーザに遊んでもらいアンケートを記入してもらい，その結果で，今回のゲーム開発の成功・失敗を判定することにした．

2.4.1. 実施内容

アンケートは下記の方々にご協力をいただいて，実際にプレイしてもらい，アンケートの記入をしていただいた．

- 大岩研究室の方々
- 横浜市立盲学校の方々
- 株式会社ユードーの方々

2.4.2. 成功目標

成功目標としては，アンケート項目の「ゲームを遊んでみて，面白い・もう一度プ

レイしてみたいと思う人の割合が，前期の『さうんど おんりぃ』プロジェクトの評価を超えること，を目的とし以下の目標を設定した．

- 面白いと思う人が 7 割以上
- もう一度やりたいと思う人が 8 割以上

これは前期での「映像のないゲーム」の最終アンケートでの結果

- 面白いと思う人が 5 割
- もう一度やりたいと思う人が 7 割

を超えることを目的として設定をした．

2.4.3. アンケート結果

アンケートを実施した結果は下記の通りである．

- 面白いと思う人が 8 割 3 分
- もう一度やりたいと思う人は 10 割

目標と設定していた回答を上回ることができ，プロジェクトは成功ということになった．ただ，残念なことに「非常に面白い」という回答を得ることが出来なかったというのが非常に残念ではあるが，まだまだ改善の余地はあるので，今後，改善を重ねていけば挽回できると思う．

また，アンケート項目のなかに，「面白いと感じた部分」を聞く項目があったが，この結果が，大岩研と盲学校で違いが現れた．大岩研の方々が，音に臨場感があるというところを面白く感じているのに対し，盲学校の方々はそれほどまで臨場感があるようには感じていないようだった．

そしてこのゲームの一番の肝となる部分であるため，場の臨場感をいかに出すかが，まだ研究，改善が必要ということの結果だと思う．

2.5. 総合評価

まず，プロジェクトの目的であるユーザが繰り返し遊んでくれるような面白いと感じられるゲームを開発することはできたのではないのかと思う．

それは，アンケート結果での目標値を大きく上回ることができたことで達成ができた.
また，プロジェクト進行，ゲーム製作を勉強していくという目標についても，ゲームを完成まで進むことができたことで十分な達成だと思える.

目標の達成に十分なほどの結果を得られたと思う.

第3章. α版開発

3.1. 開発方針

今回のプロジェクトでは，前回の「さうんど おんりぃ」プロジェクトで得られたゲーム作成の方法や技術的なノウハウを活かし，より楽しく遊ぶことのできるゲーム開発を目指した．ただし，前回のゲームの企画を元に開発するわけではなく，ゲームの企画や内容については一新し，今までにないような斬新で面白いゲームを開発することを目標とした．
そこで，ゲーム開発期間の前半は，主にゲームの企画を出すことに費やし，企画が確定した段階で本格的に実装を行い，ゲームを作り込んでいくという方針で進めることにした．

3.1.1. 開発の流れ

今回のプロジェクトでは，ゲーム開発のフェーズを大きく二つのフェーズにわけた．前半のフェーズでは，α版の開発を行い，後半のフェーズではβ版の開発を行った．
α版は，斬新なゲームのアイデアを模索したり，ゲームの方向性を確認したりするために開発する試作版である．そのため，α版では，ゲームに多少のバグや不具合があっても，遊ぶのに支障が出ない程度であれば許容するという方針で開発した．また，ゲームの完成度についても高いものは求めず，ゲームの大まかなコンセプトや雰囲気がつかめる程度のものであればよいということにした．α版の開発で，企画がほぼ固まった段階で，β版の開発に移行する．
β版は，ゲームの内容を評価していただくことを目的とした，完成品に近いものである．そのため，β版では，ゲームの根幹に関わるような重大なバグは全て解消し，完成度についても正式版の機能を一通り備えた，クオリティの高いものを完成させるという方針で開発した．

3.1.2. ミニゲームの量産

α版の開発では，斬新で面白いゲームのアイデアを出すために，また，ゲームの開発に慣れる，ゲーム開発のイメージをつかむために，手軽に開発できる規模のミニゲームを多数開発することにした．ミニゲームは，基本的にチームメンバーが個々に別々のアイデアを出し，別々に開発するという方法を採った．

出来上がったゲームはチームメンバー，または研究室の方々にプレイしてもらい，面白かった点は残す，あるいは強化し，不満や要望があれば改善しながら，また新しいゲームを開発する．そして，ある程度ゲームの方向性やコンセプト等が固まった段階で，最終的に一つの作品にまとめ上げるという方法を採ることにした．

3.1.3. ゲーム特有の開発手法

今回の開発は，PM がゲーム業界の方ということもあり，一般的にゲーム業界で行われている開発方法に従って行うことにした．

企画書については，ゲーム開発が進むにつれて新しいアイデアの追加や修正を行う必要性が生じるので，最初の段階かつ詳細で綿密な企画書を書いてもあまり意味がない．そのため，企画書は概要を記述する程度にとどめ，開発が進み，企画に変更が生じた段階で随時書き直していくという方針で進めることにした．

設計書については，企画に変更が生じ，そのたびに設計書を書き直すのは非効率であり，また，設計書自体の必要性も低いため，設計書は作成しないことにした．ただし，設計自体を行わないわけではなく，設計についての議論はチーム内で行い，チームメンバー全員が設計の内容については共有していた．また，後々の混乱を招くことがないように，重要な事柄はソースコード内にコメントとして記述しておくよう心がけた．

3.1.4. ミニゲーム企画立案

ミニゲームを作成するにあたって，まずゲームの企画を思いつく限り挙げていき，その中から面白そうなものを選び，実装していった．

企画立案の段階で挙がっていたものとしては，以下のようなものがあった．

- 旗揚げゲーム
  - 「右上げて」，「左上げて」，「右下げて」，「左下げて」といった音に合わせて，左右のキーを押して旗揚げゲームをするゲーム．
  - ゲームとしては面白そうだが，ゲームの幅を広げるのが難しそうで，単純になりそうであるという欠点があった．また，音の鳴る方向を変える必要がないので，サラウンド音声を活かすのが難しそうであるという問題もあった．

- 落ちてくる障害物をよけるゲーム
  - 複数の音が上から下に移動してきて，それらの音にあたらないように移動してよけるゲーム．音にあたってしまうとゲームオーバーとなる．ゲームのプレイ時間

が長くなるほど，音の数が増え，難易度が上がる．

- リズムゲーム
  - 上下左右から，様々なタイミングで音が鳴ってくるので，タイミングよく音のなる方向のキーを押すゲーム．タイミングが正確なほど高得点が得られ，方向を間違えたり，タイミングがずれたりするほど，得点が低くなる．

- モグラ叩き
  - 様々な方向から，モグラの鳴き声が聞こえてくるので，聞こえたところのモグラを叩いていくゲーム．
  - どのようにして，鳴いているモグラを選択するのか等が難しそうであるという問題がある．

- サウンドノベル
  - ストーリーを音声で読み上げ，ストーリーの途中で選択肢が現れ，どの選択肢を選ぶかによってストーリーが変化していくゲーム．
  - 使用する音は主に声だけなので，サラウンド音声を使う必要性が少なそうであるという問題がある．

- スポーツゲーム
  - 音のみでボウリングやテニス，サッカーやボクシング等のスポーツを再現するゲーム．
  - 音のみでどこまで現実のスポーツに近い臨場感を出せるかが課題となりそう．

- 鬼ごっこ
  - 音を頼りに鬼を追いかけ，鬼を捕まえるゲーム．
  - 前期の「さうんど おんりぃ」に近い

- 聖徳太子ゲーム
  - 一度に複数の音声が同時に読み上げられ，それぞれの音声で何をしゃべっていたのかを当てるゲーム．
  - 使用する音声は人の声だけなので，サラウンドを使う必要性が少なそうであるという問題がある．

- 記憶ゲーム
  - いくつかの音が様々な方向から聞こえてくるので，その音がどちらから聞こえて

きてかを当てるゲーム．正解数が増えるごとに記憶しなくてはならない音が多くなっていき，難易度が上がる．

この中から，チームメンバー各自が面白そうだと思うものを選び，それをベースにミニゲームを開発することにした．メンバー内で相談した結果，最終的に橋山は記憶ゲーム，藤原は聖徳太子ゲームをそれぞれ選び，開発を行った．

3.1.5. Java による小規模ゲームの開発

今回のゲーム開発では，5.1ch のサラウンド音声を使用する．しかし，サラウンド音声を扱うためのライブラリが Java にはなく，C++用のものしかなかったため，開発言語は C++を使用するということに決定した．ただし，チームメンバーは C++によるソフトウェア開発の経験がなく，慣れていなかったため，当初はチームメンバーが慣れ親しんでいる Java を使用してミニゲームを開発することにした．

ミニゲームの開発は，チームメンバーがそれぞれ別々のものを個別に開発することにした．以下，チームメンバーがそれぞれ開発したゲームについて説明する．

- 方向当てゲーム（記憶ゲーム）　担当：橋山

  方向当てゲームは，上下左右のいずれかの方向から音を流し，その音がどの方向から聞こえたかを記憶していくゲームである．正解すると，直前に流れた音に新しい音が一つ追加され，記憶する音の数が増えていく．間違えたところでゲームオーバーとなり，いくつまで正解できるかを競うゲームである．

  このゲームは Java で開発したため，サラウンド音声用のライブラリを使用することができなかったので，初めから定位や音量を決めた音を方向ごとに用意し，それらを流すという方法を採っていた．しかし，サラウンドを使用しないとどうしても音の鳴っている方向がわかりづらくなってしまい，ゲームとしてはプレイしづらいものになってしまった．

  また，音が上下左右の固定した場所から聞こえてくるだけでは単調で飽きてしまうので，音の聞こえてくる方向を移動させるといった要素があると面白くなるのではないかという意見も出た．それをふまえて，次はサラウンド音声を活かして，音の移動を取り入れて新しいゲームを開発することとなった．

- 方向当てゲーム（記憶ゲーム）の規模

  方向当てゲーム（記憶ゲーム）の規模については，以下の通りである．

ファイル形式	行数	コメント	空行	Logic	ファイ￥ル名
java	162	28	22	112	MemoryGame.java
java	62	12	8	42	MidiPlayer.java
java	65	12	13	40	Sound.java

ファイル形式	行数	コメント	空行	Logic	ファイル数
java	289	52	43	194	3
合計	289	52	43	194	3

- 聖徳太子ゲーム　担当：藤原

聖徳太子ゲームは，同時に複数の果物の名前を読み上げ，それを全て聞き分けるゲームである．読み上げられた名前はキーボードで入力し，それが全て合っていたら正解となり，新しい問題が出題される．また間違っていたら，ゲームオーバーとなる．

このゲームは，内容としては面白いが，キーボードを使った入力が大変であるという問題があった．例えば，タッチタイピングができない人にとっては，正しい文字を入力するのが困難であるという問題がある．また，一度入力した文字は修正ができない等の不具合もあった．

また，この時点では読み上げる果物の音声に，機械的な合成音声を使用していたため，イントネーションが不正確で，聞き取りづらいという問題があった．さらに，全ての音声が同じ位置から聞こえてくると，言葉が混じって聞こえてしまうため，それぞれの音声は別々の位置から聞こえるようにした方がよいという意見が出た．それを踏まえて，次はこのゲームの基本ルールは変えずに，より遊びやすいものを開発することとなった．

- 聖徳太子ゲームの規模

聖徳太子ゲームの規模については，以下の通りである．

ファイル形式	行数	コメント	空行	Logic	ファイル名
java	81	26	11	44	Input.java
java	133	20	7	106	SoundManager.java
java	124	24	11	89	SyotokutaishiGame.java

ファイル形式	行数	コメント	空行	Logic	ファイル数
java	338	70	29	239	3
合計	338	70	29	239	3

3.1.6. C++による小規模ゲームの開発

一度目のミニゲーム作成では，普段から慣れている Java で実装を行い，手っ取り早く簡単なゲームを開発し，ゲームのイメージをつかんだ．それをふまえて，二度目のミニゲーム作成は，C++のサラウンド音声ライブラリを使用して行った．

二度目のミニゲーム作成も，チームメンバーがそれぞれ別々のものを個別に開発することにした．以下，チームメンバーがそれぞれ作成したゲームについて説明する．

- さうんどシュート　担当：橋山

  さうんどシュートは，音が左から右に移動しながら鳴り続けており，自分の正面に音が来たと思ったらキーを押し，その精度を競うゲームである．

  最初は左の方から音が鳴っており，音量も小さいが，だんだんと音が右に移動していき，音量も大きくなってくる．音が自分の目の前を通り過ぎると，音は自分の右側に遠ざかっていき，音量も小さくなっていく．音が自分の目の前に近く，音量が大きいときにキーを押すほど高いスコアを獲得でき，音が自分の目の前から遠く，音量が小さいほど，スコアは低くなるというゲームである．

- さうんどシュートの規模

  さうんどシュートの規模については，以下の通りである．

ファイル形式	行数	コメント	空行	Logic	ファイル名
h	32	0	7	25	SoundContainer.h
h	39	3	4	32	SoundPlayer.h
h	14	1	1	12	StateMessages.h
h	38	3	9	26	TitleState.h
cpp	57	16	6	35	Game.cpp

cpp	307	64	52	191	GamePlayState.cpp
cpp	44	15	4	25	KeyFlag.cpp
cpp	65	21	9	35	KeyState.cpp
cpp	45	6	9	30	Main.cpp
cpp	85	24	13	48	OnMapObject.cpp
cpp	91	17	16	58	SoundContainer.cpp
cpp	159	6	21	132	SoundPlayer.cpp
cpp	114	25	21	68	TitleState.cpp
h	23	0	5	18	Game.h
h	58	0	9	49	GamePlayState.h
h	18	0	3	15	KeyFlag.h
h	33	0	7	26	KeyState.h
h	64	4	14	46	OnMapObject.h
h	36	1	4	31	SDLCommonFunctions.h

ファイル形式	行数	コメント	空行	Logic	ファイル数
cpp	967	194	151	622	9
h	355	12	63	280	10
合計	1,322	206	214	902	19

● 聖徳太子ゲーム改　担当：藤原

聖徳太子ゲーム改は，Java で開発した聖徳太子ゲームをベースにして，読み上げられる音声が様々な位置から流れてくるようにしたものである．

Java で作成したゲームは，音の定位が一定だったため，音声が聞き取りづらいという問題があったが，C++版では，サラウンド音声を使用して，それぞれの音声が別々の位置から聞こえてくるように改良したため，Java で作成したものよりも音声が聞き取りやすくなっている．

● 聖徳太子ゲーム改の規模

聖徳太子ゲーム改の規模については，以下の通りである．

ファイル形式	行数	コメント	空行	Logic	ファイル名
h	39	3	4	32	SoundPlayer.h
h	14	1	1	12	StateMessages.h
h	35	3	6	26	TitleState.h
cpp	57	16	6	35	Game.cpp
cpp	365	69	55	241	GamePlayState.cpp
cpp	44	15	4	25	KeyFlag.cpp
cpp	65	21	9	35	KeyState.cpp
cpp	45	6	9	30	Main.cpp
cpp	85	24	13	48	OnMapObject.cpp
cpp	91	17	16	58	SoundContainer.cpp
cpp	159	6	21	132	SoundPlayer.cpp
cpp	114	25	21	68	TitleState.cpp
h	23	0	5	18	Game.h
h	66	0	8	58	GamePlayState.h
h	18	0	3	15	KeyFlag.h
h	33	0	7	26	KeyState.h
h	64	4	14	46	OnMapObject.h
h	36	1	4	31	SDLCommonFunctions.h
h	32	0	7	25	SoundContainer.h

ファイル形式	行数	コメント	空行	Logic	ファイル数
cpp	1,025	199	154	672	9
h	360	12	59	289	10
合計	1,385	211	213	961	19

3.2. ミニゲーム評価

3.2.1. クライアント評価

本プロジェクトのクライアントである株式会社ユードーの南雲玲生代表取締役に，ミニゲームについての意見をいただいた．

南雲代表取締役に指摘していただいた点は，主に以下の二点であった．

- 音の質にこだわるべきである

  我々が開発したゲームは，どれも音声に合成音声を用いており，機械的であじけないものになってしまっていた．映像を使わない音だけのゲームにおいては，ゲームの面白さや迫力，魅力は音の質によって左右される部分が非常に大きいというお話しをいただいた．特に，電子的な音は，人間の耳にとって非常に不自然で耳障りであるため，今回のゲームで使用する音は，全て自然の音を使用した方がよいだろうとのことであった．

  例えば，音声は実際に人がしゃべった声を録音する，とりわけ女性の声にするとよいのではないかという意見や，必要な音はマイクを持って出かけていき，自分たちで録音するとよいのではないかという意見をいただいた．また，どうしても手に入らない音や，手軽に手に入らない音があった場合は，南雲代表取締役が相談に乗ってくださるというお話しもいただいた．

- 今までにないような斬新なゲームにして欲しい

  また，我々が作成したゲームは，内容がありきたりで，斬新さに欠けるものであった．そのため，インパクトが弱く，飽きるのも早くなってしまうという欠点があった．

  南雲代表取締役からは，せっかく「映像を用いないゲーム」という新しい試みに挑戦しているのであるから，ゲーム自体の企画についてもより斬新で，今までに例を見ないようなものを作成した方が，作り甲斐もあり，プレイする側としても楽しめるのではないか，また，ゲームが投げかける影響力も大きいのではないかという意見をいただいた．

  例えば，商店街の中を散歩するゲームなどの案を提示していただいた．自分たちで商店街の中を歩きながら商店街の音を録音し，その音をもとに実際に商店街に出かけている様子を音だけで再現でき，さらに商店街の移りゆく音を環境音楽や BGM としても楽しめるようなゲームにすれば，既存のゲームの枠を取り払った，全く新しいジャンルを確立できるのではないかという意見をいただいた．

上記のようなクライアント評価の結果をふまえ，次の開発では南雲代表取締役からいただいた意見を参考に，音の質にこだわり，より斬新なゲームを作ることを目標にした．

3.2.2. **中間報告会**

中間報告会では，評価委員会の方々から大変的確かつ有用な意見や指摘をいくつかいただくことができた．中間報告会でいただいた意見は，主に企画について，もう少し洗練された，面白いものを考えてみて欲しいというものであった．また，具体的にいくつかアイデアを出していただいた．評価委員会の方からいただいたアイデアは，以下のようなものである．

- 絶対音感育成ゲーム

  絶対音感育成ゲームは，音を鳴らして，その音の音程を当てるゲームである．ゲームをプレイしているうちに，絶対音感を身につけられるようなものだと，面白そうだし需要も高いのではないかという意見をいただいた．

- 音キムス

  音キムスは，あるものが振動する音を鳴らし，それが何の音であるのかを当てるゲームである．例えば，小豆やビー玉等の入った容器を振ったり動かしたりしたときの音を鳴らし，それらの音の正体が何なのかを正しく判別し，当てるゲームである．

  臭いを当てる臭いキムス等は，すでにあるそうだが，音キムスは音の大きさや中身に使うものの準備など色々と問題が多く，ゲームとして遊ぶのはなかなか難しいとのことである．そのため，音が準備されていて音の大きさが自由に変えられるゲームであれば，より簡単に音キムスが実現できると思われるので，今回のプロジェクトでチャレンジしてみるのも面白いのではないかという意見をいただいた．

- ファンタジックアドベンチャーゲーム

  ファンタジックアドベンチャーゲームは，映像での描写が難しい世界を，音だけを使って体感できるようなゲームである．例えば，ハリーポッターの魔法の世界などを音だけで再現し，実際に冒険しているような気分が味わえるようなゲームである．

- バーチャルシミュレーションゲーム

  バーチャルシミュレーションゲームは，現実世界で行われていることを音だけでシミュレートして体感できるようなゲームである．例えば，音だけを使って電車の運転を疑似体験できるような，音だけでプレイできる電車で GO のようなゲームである．

  実際に視覚にハンディキャップを有する方々の間では，電車で GO は人気が高いらしく，バーチャルシミュレーションゲームならば，視覚にハンディキャップを有する方々にも楽しく遊んでもらえるのではないかという意見をいただいた．

  我々がβ版で開発することにした虫捕りゲームも，このバーチャルシミュレーショ

ンゲームに近いものである．虫捕りゲームについての詳細は後述する．

また，中間報告会でいただいた質問の内容は，主に以下の 2 点であった．

- 前期のプロジェクトよりも優れている点，進歩した点は何か？

今回の「さうんど おんりぃ 2」プロジェクトは，前期の「さうんど おんりぃ」プロジェクトの続編としてスタートしたものであり，映像を用いないゲームを開発するという目的については共通だった．中間報告会では，前期プロジェクトとの違いに関する説明が欠けていたため，誤解や疑問を生じさせる結果となってしまった．

この質問に対する答えとしては，次のようになる．前期のプロジェクトはサラウンド音声を使ったゲーム作成を行うための技術調査を行うことがメインであり，ゲームの品質についてはあまり高いものを求めていなかった．一方，今期のプロジェクトでは，前期のプロジェクトを通して得られた知識や技術，ノウハウなどを活かし，前期のプロジェクトよりも面白く，完成度の高いゲームを作成することを主眼においており，この点が前期のプロジェクトと異なる点，進歩したといえる点である．

しかし，中間報告会で発表したゲームはクオリティが低く，面白みにも欠けていたため，今期のプロジェクトではゲームの品質を重視するという点をアピールすることができなかった．改めて，ゲームアイデアの抜本的見直しと，音質向上の必要性を認識させられた．

- PM は何をマネージメントするのか？

質問を受けた時，明確な回答を持って回答することが出来なかった．そのことで，私自信，PM のことをまるで理解をしていないということを痛いほど感じ，考えさせられた．

この質問について改めて回答すると，PM とは，プロジェクトの計画とそれに対する現実との差異を認識し，プロジェクトの最終ゴール達成のために，継続的に，適切な軌道修正を行っていくことが PM としての役割である．

今回のプロジェクトでは，面白いゲームを作成する（プロジェクトの最終ゴール）のために，何が面白いのか，そのアイデアで，本当に面白いのかなど，ゲームとしての面白さの修正（プロジェクトの軌道修正）を行っていくのが，私が PM としてのマネージメントだと考える．

3.3. ミニゲーム開発を踏まえて

これまではいくつものゲームを作っては壊すということを繰り返した．それによって「映像を用いないゲーム」を作るための全体的なイメージは掴めたものの，実際に出来上がったゲームは自分たちで遊んでみても正直つまらないものばかりであった．ゲーム開発にお

いて，開発したゲームが面白くないということはその開発の失敗を意味する．そこで，我々は今後の開発において同じ失敗を繰り返さないように，ミニゲームの開発で失敗した原因とそこから得られた反省点について考察した．

3.3.1. 質の高い音の重要性

前回の「さうんど おんりぃ」にも言えることだが，ミニゲームを作る時に我々は動くものを作ることを優先した．表現力が乏しい電子音や合成音声を多用して，音にはまったくこだわらなかった．これは，我々が普段遊ぶ一般的なゲームがイメージとしてあったからである．

映像があるゲームでは，映像が多くの情報を持っており音は副次的な要素である．音自体に情報が少なくても映像でカバーすることが出来るため，結果として面白いゲームになることがありうる．しかし，我々が作っている「映像を用いないゲーム」は文字通り音がすべてである．そのため，ゲーム中の音が表現力に乏しい電子音や合成音声では，そこからゲーム全体をイメージとして捉えることが非常に困難になる．

例えば，ゲームの内容を説明する音声が合成音声であるときと人間の生の声を録音したものであるときを比較してみるとよく分かる．現在無料で利用できる合成音声は，記述した文章を読み上げるといったタイプのものだが，これは基本的に文章を棒読みする．また，文章全体を読ませるとほぼ間違いなくイントネーションが不自然となる．合成音声の説明ではゲームの内容は伝わるかもしれないが，平坦な音声ではゲームの重要な部分が曖昧になってしまう恐れがある．これが人間の生の声であれば，アクセントやイントネーション，微妙な間などを上手に使うことで，どの情報を一番伝えたいのかということを明確にすることができる．ゲームの設定が近未来でロボットが説明しているというような特殊な状況下でない限り，合成音声では人間の生の声に比べて情報がうまく伝わらないのである．

以上のことから我々は，「映像を用いないゲーム」を作る際にはまず質の高い音を収集し，それらを使ってゲームを開発していくべきである，という結論に至った．

3.3.2. 企画に対するモチベーション

ミニゲームを開発するにあたり企画自体はいくつか挙がったものの，どれも似通ったもので自分たちが面白いと思う企画を考えることが出来なかった．これはメンバーに企画力が足りなかったことが原因である．そのような企画を基に開発を進めていったため，当然納得できるような面白さを備えたゲームは出来なかった．

しかし，ゲームを開発しているうちに研究室にいる人や評価委員の方から企画に対する様々なアドバイスを頂いた．中にはとても面白そうな企画を持っている人がいて，是非そ

れを作って欲しいとお願いされたこともあった．そういった中で様々な企画に触れているうちに，自分たちだけで企画を考えようとしていたことが失敗の原因であるということが分かった．我々は企画を実装に結び付けて考えてしまうので，実現可能性や実装の難易度が頭にあるためどうしても視野が狭くなってしまう．しかも，メンバー内でしか企画を話し合っていなかったので，ますますその傾向は強くなっていったのである．

我々には「映像を用いないゲーム」という未知のジャンルに対して，面白い企画を持っている人はたくさんいるという認識が欠けていた．開発者という立場に縛られない自由な発想をする外部の意見や企画を取り入れることで，より魅力的な企画が出来上がるのだということを改めて発見した．

3.3.3. 新しい企画の模索

以上のことを活かし，企画力に定評のある大岩研究室の荒木氏から，色々と意見やアイデアを提供してもらい，それを参考に企画を考えるという方法を採ることにした．荒木氏からは，新鮮で率直な発想と，いくつかの大変興味深いアイデアを提供していただいた．例えば，以下のようなものである．

背景：

荒木氏には，視覚にハンディキャップを有する友達がいて，その友人に子供が生まれた．そこで，健常者の子供と視覚にハンディキャップを有する親が全くストレスを感じずに遊べるゲームがあるとよいのではないかと考えた．視覚にハンディキャップを有する人は，音を立体的に捉えることができるので，その感覚を追体験できるようなゲームがあると面白いのではないだろうか．

現実的には，健常者と視覚にハンディキャップを有する人が一緒に遊ぶことは難しいが，音だけの世界であれば，一緒に遊ぶことが可能である．音を介して，一緒にどこかに出かけたり，遊んだりすることが，何の抵抗も無く楽しむことができる．現実にある楽しい時間を，視覚にハンディキャップを有するか有しないかに関わらず楽しむことができる．そこで，森の中や海の中や街の中，あるいは現実には体験できない，コンピュータの中や身体の中などの場所を移動し，そこで一緒に遊べるようなゲームを作ってみると楽しいのではないだろうか．

具体案：

日常的で，臨場感のあるゲーム：

- 虫捕り
  - 市販されているゲームである「僕の夏休み」のように，季節感や自然の雰囲気が体感できるようなものは，健常者にとっても面白く，魅力的である．虫捕り

という内容ならば，様々な季節や時間の移り変わりが再現でき，森に生息する様々な生き物の存在が感じられるので，面白いのではないだろうか.

- 焼肉
  - 視覚にハンディキャップを有する方は，鉄板がどこにあるかわからないため火傷をしてしまう可能性があるので，焼き肉を食べにいくことができないという.そこで，音で焼き肉を再現し，焼き肉の楽しさを体験できるようなものができたら嬉しいのではないだろうか.

- オリエンテーリング
  - 夏のキャンプに参加し，クイズに答える等.

- 特定の人物を捜すゲーム
  - 人混みの中で迷子になった子供を捜す等.

- 料理
  - 音だけで料理の臨場感が味わえるようなゲーム.

- 釣り
  - 音だけで釣りの臨場感が味わえるようなゲーム.

- 肝試し
  - 音だけで肝試しの臨場感が味わえるようなゲーム.

対戦要素のあるゲーム：
- 雪合戦
- スイカ割り
- ビーチバレー
- ピンポン（卓球）
- 蚊やハエを叩く
- 射的

現実では体験できない世界をシミュレートするようなゲーム：
- 人間の体内を探検する
  - 自分が小さな医者になって，病原菌と戦うことができる.

- 宇宙空間で敵と戦う

- 人間以外の生き物になって，人間以外の生き物の世界を疑似体験する
  - 野良猫になって，夜に人間から食べ物を奪う等.

以上のような案の中から，チームメンバーが最も興味を示し，最もアイデアの膨らんだ虫捕りゲームの案を採用した．この虫捕りゲームのアイデアを基に再度ゲームを作り直し，ある程度作成してみて面白くなりそうであれば，企画を虫捕りゲームに絞り，α版最後の開発とすることにした.

3.4. 虫捕りゲーム開発

α版として最終的に企画がまとまった虫捕りゲームについて，開発方針を決めて，企画書を作成した.

3.4.1. 企画書の作成

「虫捕り」とキーワードをベースにメンバー間で議論した結果,「森の中を散歩しながら虫を捕まえていく」という基本的なゲームの企画が出来上がった．最初に作った企画書は以下の通りである（図 2-1～2-5 参照).

図 2-1　虫捕りゲーム企画書（1 ページ）

図 2-2　虫捕りゲーム企画書（2 ページ）

図 2-3　虫捕りゲーム企画書（3 ページ）

ゲームの操作

- ↑：前進する
- ←→：向きを変える
- Enter：網を振る（虫を捕まえる）
- Esc：ゲーム終了

4

図 2-4　虫捕りゲーム企画書（4 ページ）

音の動きかた

- BGM
  - ゲームの雰囲気（森の音など）
  - ループし続ける
  - プレーヤーの動きによって音量・定位が変化しない
- オブジェクト
  - ゲーム内で動いたり音を発したりしているもの
  - 川の流れ，鳥・虫の鳴き声，風の音など
  - プレーヤーの動きによって音量・定位が変化する

5

図 2-5　虫捕りゲーム企画書（5 ページ）

本プロジェクトでは企画書が仕様書を兼ねるため，基本的な企画に加えてゲームの流れや操作方法，登場するゲーム要素に対しての大まかな仕様と，シーンごとに必要となる音のリストなどを併せて記載した. 修正した企画書は以下の通りである（図 2-6～2-19 参照）.

図 2-6　虫捕りゲーム企画書修正後（1 ページ）

企画の概要

- 虫取りゲーム
  - 森を散歩しながら制限時間内にできるだけ多くの虫を捕まえる
  - ゲームであると同時に，自然の音に囲まれることでリラックスすることもできる

2

図 2-7　虫捕りゲーム企画書修正後（2 ページ）

図 2-8　虫捕りゲーム企画書修正後（3 ページ）

図 2-9　虫捕りゲーム企画書修正後（4 ページ）

図 2-10　虫捕りゲーム企画書修正後（5 ページ）

図 2-11　虫捕りゲーム企画書修正後（6 ページ）

図 2-12　虫捕りゲーム企画書修正後（7 ページ）

図 2-13　虫捕りゲーム企画書修正後（8 ページ）

動物

- 捕獲対象ではない
- チンパンジー (昼のみ)
  - 鳴きながらランダムに動き回っている
- 鳥 (昼のみ)
  - 鳴きながら，プレイヤーの頭上を通過する
- フクロウ (夜のみ)
  - 木の上にとまって，鳴いている

9

図 2-14　虫捕りゲーム企画書修正後（9 ページ）

自然

- 森
  - 虫を捕る場所 (空間) である
  - どこにいても同じ音量と定位で音を発する
- 川
  - 絶えず水が流れていて，音を発している
  - 川に入ると足音が変わる
- 風
  - 一定の場所を通過すると，風が吹く

10

図 2-15　虫捕りゲーム企画書修正後（10 ページ）

図 2-16　虫捕りゲーム企画書修正後（11 ページ）

図 2-17　虫捕りゲーム企画書修正後（12 ページ）

図 2-18　虫捕りゲーム企画書修正後（13 ページ）

図 2-19　虫捕りゲーム企画書修正後（14 ページ）

3.4.2. 開発方針の確認

ミニゲームの開発ではいくつものゲームを作って，そこから面白そうな要素を抽出してひとつのゲームを作り上げるという予定だった．ミニゲームの開発で作られたゲームの中には面白さが期待される要素を含んだゲームはあるにはあったが，決定的なものではなかった．そこで方針を変更し，研究会同期の荒木氏が提案した数ある企画の中でメンバーが興味を持った虫捕りという要素をベースに開発することにした．

ミニゲームの開発を踏まえて，虫捕りゲームを開発する際には以下の点に留意した．

- 虫捕りのイメージを喚起するような良質の音を集めることを優先する
- 企画に詰まったら積極的に他人に相談し，魅力的な意見を取り入れていく

以上の開発方針を踏まえて，次のような実装手順で開発を行った．

1. 音環境の作成
   - 良質な音を収集する
   - 虫捕りを行う場所である森を作る
   - 森の中を散歩できるようにする
2. ゲームの骨格部分の作成
   - 虫捕りを行えるようにする

3.4.3. 音環境の作成

まずは，ゲームの要となる音を探すところから始めた．知り合いや大学の図書館から効果音を集めた CD を借りたり，クライアントである南雲氏に使えそうな音をもらったりした．音の中にはそのまま使えないものもあったので，フリーの音声加工ツールである SoundEngine（http://www.cycleof5th.com/download/）を利用して，適切な加工を施した．

音がある程度揃ったところで，いよいよコーディングに入った．音を鳴らす部分については，前回の「さうんど おんりぃ」プロジェクトで利用した SoundPlayer と SoundContainer というクラスがあったので，これを利用することで難なく実装できた．これらは，ミドルウェアである CRI Audio の中の CriAuPlayer をラップしたもので，音の再生や停止を管理している．BGM として森の雰囲気を出す音をループ再生しておき，その中で虫や動物の音が点在するようにした．同時に，デバッグのためにゲーム中のプレーヤーや動物などの位置関係が把握できるように簡単な描画が行うようにした．

次に行ったのはプレーヤーが散歩できる，という部分である．前回の「さうんど おんりぃ」ではプレーヤーは上キーを押したら上に，左キーを押したら左へ移動するというカニ歩きの移動方法を採用していた．しかし，今回はリアリティを追及するために，左右キー

を押したら自分の向きを変えるという移動方法を採用することにした．

プレーヤーの移動方法を決定したところで，次に音の聞こえ方について考えた．今回採用した移動方法では，プレーヤーが向きを変えるとそれに応じて音の聞こえ方を修正しなければならない．それを実現するために行った実装と数式を以下に記す．

1. プレーヤー（x，y）と，音を鳴らしている対象（x_1，y_1）との角度θを計算する
   $\theta = \mathrm{atan2}(x_1 - x, y_1 - y)$
   ※原点から引数に与えられた座標までの角度をラジアンで返す atan2 関数を利用した．

2. プレーヤーと，音を鳴らしている対象との距離 d を計算する
   $d^2 = (x_1 - x)^2 + (y_1 - y)^2$

3. プレーヤーが変えた向きαと距離 d をもとに，対象の新座標（x_2，y_2）を計算する
   $x_2 = d * \cos(\theta + \alpha) + x$
   $y_2 = d * \sin(\theta + \alpha) + y$

4. 新座標を元に，各スピーカーに送る音の大きさを再計算して設定する
   ※この処理は，CRI Audio 側の関数を利用した

また，虫や動物などプレーヤーとの角度や距離によって音の聞こえ方が変わるものは BGM と分ける必要があると考え，MovableSound というクラスを新たに作成した．この時点でのクラス図が図 2-20 である．GamePlayState がゲームそのものであり，音として SoundMaterial と MovableSound を持っている．GamePlayState は MovableSound の moveSound()メソッドを逐次呼び出し，その度に各オブジェクトがプレーヤーとの距離や角度を再計算する．ここで再計算された情報は再生中の音にも適用されるので，音の定位や音量がリアルタイムで変化し続けるのである．

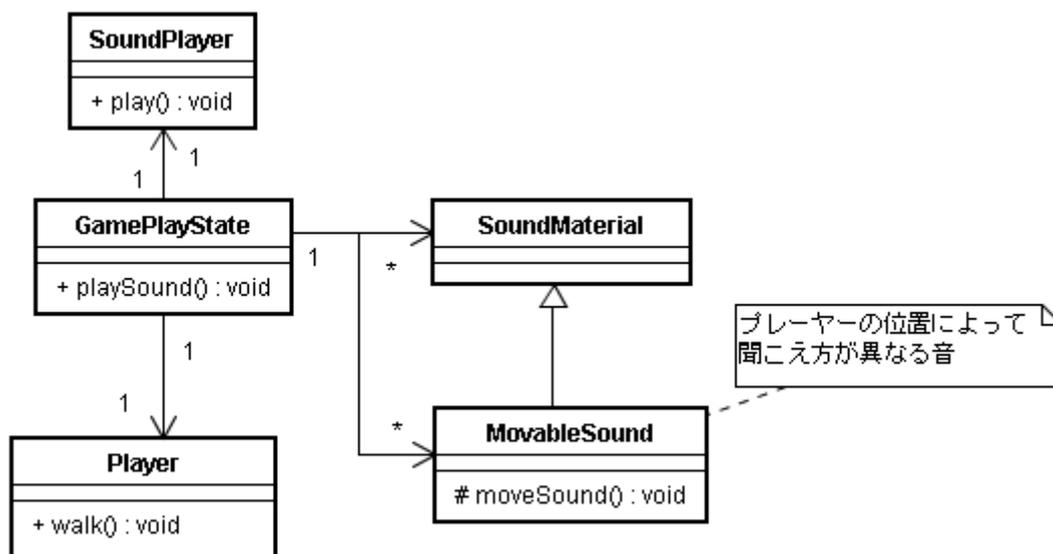



図 2-20　音環境作成時のクラス図

最後に，音量について述べておく．まずは，表 2-1 を見てもらいたい．この表はプレーヤーと音を鳴らす対象物の距離と音量を表している．青い線が改善前の音量の計算式を，赤い線が改善後の音量の計算式をもとに描いた距離と音量の関係である．

- 改善前の式：(音量) = 1 / sqrt(距離) * 2
- 改善後の式：(音量) = －sqrt(距離) / 10 + 2

※sqrt は平方根を表す
※音量の単位はリニアスケールである（1.0 が本来の音量）

グラフを見ると分かるように，改善前の式では距離が離れているうちは音量も緩やかに大きくなっていくが，距離が 50 を切ると音量は急激に大きくなる．これは，自然界で実際に聞こえる距離と音の関係を数式化したものである．しかし，実際にこの式を用いてゲームを行ってみると音量の変化が乏しく，距離が近づいても判断することが非常に困難であることが分かった．そのため，我々はゲームにふさわしい音量の計算式を改善した．それが改善後の数式である．

改善後の数式では，改善前の数式に比べて距離が離れていても音量が変化する様子が分かるようになった．また，距離が一定以上開くと音が完全に消えるため，音の種類や数が増えても煩雑さが減って聞き取りやすくなった．

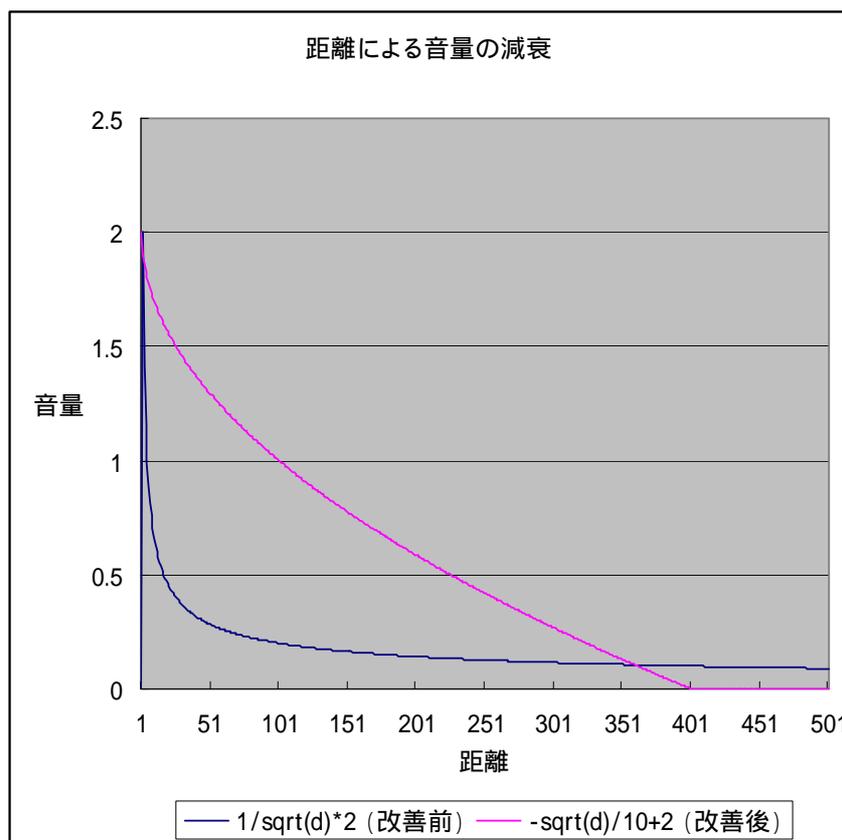



表 2-1 距離による音量の減衰

3.4.4. ゲームの骨格部分の作成

虫を捕獲する機能を追加する前に，MovableSound クラスを継承して，Creature クラスを作った．さらに Creature クラスを継承して Insect クラスを作った．これは，ゲーム中に存在するオブジェクトの持つ役割を明確に分けるためである．ここで，現在の音オブジェクトについて整理しておく．

- SoundMaterial…ゲーム中に存在する音オブジェクトすべてを表す
- MovableSound…プレーヤーの座標によって音の聞こえ方が変わるもの（川･泉など）
- Creature…MovableSound のうち，自ら移動するもの
- Insect…Creature のうち，プレーヤーが捕まえることが出来るもの

この時点で，クラス図は図 2-22 のようになった．

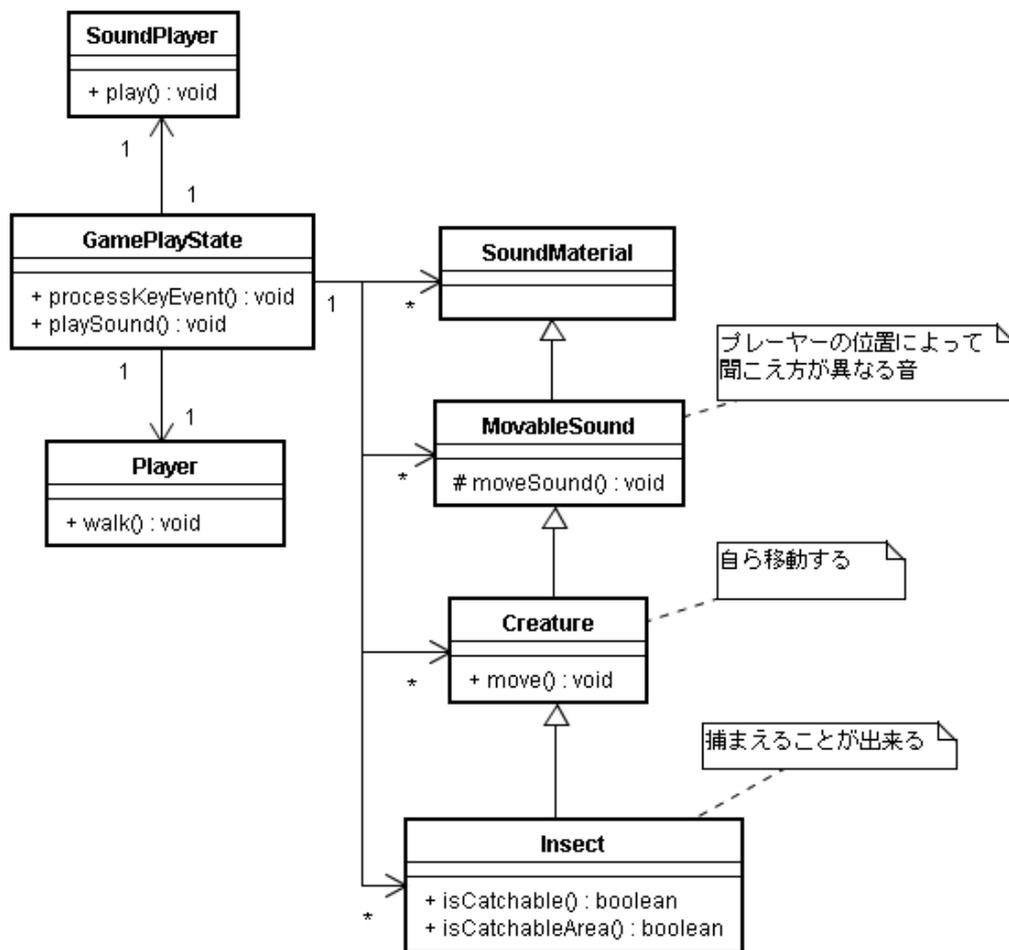



図 2-22　ゲームの骨格部分作成時のクラス図

この設計を元に，プレーヤーが網を振ると，GamePlayState が Player と Insect の距離を計算して，一定の範囲内に捕まえられる虫がいれば捕まえる，という実装を行った．

3.5.　α版評価（ユーザレビュー）

ゲームの骨格部分が完成した時点で，慶応義塾大学の中根様にプレイしていただき，レビューをいただいた．実際に視覚にハンディキャップを有する方の視点から，プレイする上での不具合や不足点，要望や意見等をいただき，ゲームの開発に反映させることで，視覚にハンディキャップを有する方々により快適に遊んでいただけるような作品にしていきたいと考えた．

中根様からは，いくつかの非常に貴重なご意見をいただくことができた．具体的には以下のような点である．

● ゲームのフィールド全体が，どのくらいの大きさか分からない．

中根様にユーザレビューを依頼した時点の虫捕りゲームでは，プレーヤーは際限なくどこまでも移動できるようになっていた．そのため，プレーヤーが遠くに移動しすぎると虫が全くいない状態になってしまい，戻る道もわからず迷子になってしまう．

そこで，移動できる範囲を定めて，移動できる範囲の限界まで移動したら，障害物などによってそれ以上移動できないことがわかるようにするとよいのではないかという意見をいただいた．

● 森の中を歩いていても，何にもぶつからない．

中根様にユーザレビューを依頼した時点の虫捕りゲームでは，木や川などの障害物は実装されていたが，障害物と同じ座標までプレーヤーが移動しても，ぶつかる音や，水の中に入る音などはならないようになっていた．そのため，実際に森の中を散歩しているような臨場感が得られず，不自然さが残ってしまっていた．

そこで，木にぶつかる音や，草を掻き分ける音などを追加すれば，よりリアルに森の中を散歩している状況が再現でき，面白さも増すのではないかという意見をいただいた．

● 足音がない

中根様にユーザレビューを依頼した時点の虫捕りゲームでは，プレーヤーが歩いても足音がしないようになっていた．そのため，自分が移動しているということがわかりづらく，位置感覚がつかみにくいという問題があった．

そこで，プレーヤーが歩いているときは足音を鳴らすようにすれば，プレーヤーが移動していることがわかりやすいし，位置感覚も把握しやすくなるのではないかという意見をいただいた．

● 川が不自然である．

中根様にユーザレビューを依頼した時点の虫捕りゲームでは，川の音が一点からしか聞こえず，泉のように聞こえてしまっていた．また，プレーヤーが川の上を通り過ぎても，何も音がしないので，不自然であるという問題があった．

そこで，川の音が一点から聞こえるのではなく，線上から聞こえるようにした方がよいという意見をいただいた．また，プレーヤーが川の上を移動したときには，なんらかの音を鳴らす，あるいは川の上は移動できないようにするとよいのではないかという意見をいただいた．

● 虫が自動的にうごく

中根様にユーザレビューを依頼した時点の虫捕りゲームでは，虫は一定の場所から動かず，止まったままになっていた．虫は動かなくても，最初からそういうものだと思ってプ

レイすれば不自然には感じないが，多少動く虫がいた方が，自然の虫捕りに近くなるのではないかという意見をいただいた.

● ただ虫を捕り続けるというだけでは飽きる

中根様にユーザレビューを依頼した時点の虫捕りゲームでは，ひたすら虫を捕り続け，ゲームの終了などもなかった．そこで，ゲームの終了条件や，虫を捕る上で目標となるようなスコアを付けた方がより楽しめ，飽きもこないのではないかという意見をいただいた．また，虫捕り以外のイベントも取り入れると面白いのではないかという意見をいただいた．例えば，虫捕りをする前に虫捕り網を探すイベントを追加したり，ゲームが進むにつれて虫捕り以外のことができるようにしたりするといった要素を取り入れることで，飽きることなく，長時間プレイできるゲームになるのではないかという意見をいただいた.

● 説明音声が足りない

中根様にユーザレビューを依頼した時点の虫捕りゲームでは，ゲームのルールや，操作方法の説明等の音声が一切入っておらず，ゲームのルールや操作方法がわかりづらいといった問題があった．そこで，ゲームのルールや，操作方法の説明を音声で付けた方よいという意見をいただいた.

音声については，人間の声で録音した方がいいが，あまり感情を込めたり抑揚を付けたりしない方がよいのではないかとのことだった．アクションゲームや RPG では，感情を込めた音声の方が，臨場感が出るが，虫捕りゲームの場合は，自然の音によるリアリティを求めた方がよいので，音声にはあまり抑揚は付けなくてもよいのではないかという意見をいただいた.

また，合成音声を使いたい場合は，ペンタックスが開発しているライブラリが，現行では一番クオリティが高いとのアドバイスもいただいた.

中根様からいただいた，これらのアドバイスを参考にして，虫捕りゲームに改良を加えより面白く，遊びやすいものにすることを目指した．またこのユーザレビューをもって，α版の開発を終了とした.

第4章. β版開発

4.1. β版実装

α版の開発を引き継いで，β版の開発を行った．主にユーザインタビューの反映とゲームとして面白くするためのギミックの追加という視点から，次のような項目の実装を行った．

1. ユーザインタビューの反映
    - 足音の実装
    - 川の音の修正
2. バグの修正
    - 虫を捕る方法の変更
    - 音の鳴らし方の変更
    - 音ファイルの読み込みにおけるバグの修正
    - 実行速度が遅い問題の解決
3. ゲームとして面白くするためのギミックの追加
    - 制限時間やスコアなどを追加する
    - 虫が簡単に捕まえられないようにする
    - 音声によって，ゲーム全体を分かりやすくする
    - タイトルやチュートリアルなどを追加する

以下では，各項目について実装の経緯や悩んだ点について説明していく．

4.1.1. ユーザインタビューの反映

ここでは，ユーザインタビューで得られた評価のうち，特に優先順位が高いと判断した 2 項目について実装・修正を行った．

- 足音の実装

これまでは，プレーヤーはただ移動するだけだったが，自分たちでゲームをプレイしてみると確かに動いたことがわからないので距離が掴みづらい．そこで，一定距離を動くごとに足音をつけることにした．

効果音 CD から足音の音を探して移動中はループ再生するようにしたが，不自然さが残っていた．その理由を考えていると，足音がずっと同じ音であることが原因になっていることが分かったので，左右の足で違う足音を用意して，歩くたびに左右の足が入れ替わるといった実装にした．すると，以前に比べて自然な足音らしく聞こえるようになった．

また，森を歩くときと川を歩くときで音を分けた．森を歩くときは落ち葉を踏んだときの音，川を歩くときは水辺を歩く音を使うことで，ゲームに臨場感が出た．

- 川の音の修正

次に川の実装であるが，今までの実装では川は点（0 次元）で表されていい．しかし，ユーザインタビューで指摘があったとおり元来川は線（１次元）である．これを解決するためには，川の端から端まで至るところで川が流れる音が同時に聞こえる必要がある．しかし，音を複数同時に再生するためにはどのように行えばよいかというライブラリ依存の技術的な問題があったため，ライブラリの提供元である CRI・ミドルウェア社に質問のメールを送った．

その結果，複数の SoundPlayer を作成することで同時再生が可能だということが分かったので，早速川における音の聞こえ方を修正した．結果として，川はそれらしく聞こえるようにはなったが，近づくと複数の川の音が干渉し合い，音が濁ってしまうという不具合が発生した．この問題については現在技術調査中であり，未解決となっている．

4.1.2. デバッグ・修正およびリファクタリング１

ユーザインタビューで得られた評価を反映した後に，一度バグの修正とリファクタリングを行うことした．これは，パソコンによって実行速度が著しく異なるためである．まず，全体のリファクタリングを行ってソースの状況を整理して，その後実行速度を落としている影響を調べるという順序で作業を行うことにした．

リファクタリングした項目はいくつかあるが，その中でも大きな変更を加えたものを記しておく．

- 虫を捕る方法の変更

従来は，GamePlayState 内で虫を捕まえるキーが押された場合，直接虫に対して捕まえられるかを調べて，捕獲範囲内にいれば捕まえるという方法を取っていた（図 3-1 参照）．しかし，プレーヤーを介さずに直接虫を捕まえると，将来虫かごを実装する場合に拡張性がなく，また意味的にも不自然である．そのため，GamePlayState 内で捕獲範囲内にいる虫をプレーヤーに知らせて，プレーヤーが虫を捕まえるように変更した（図 3-2 参照）．

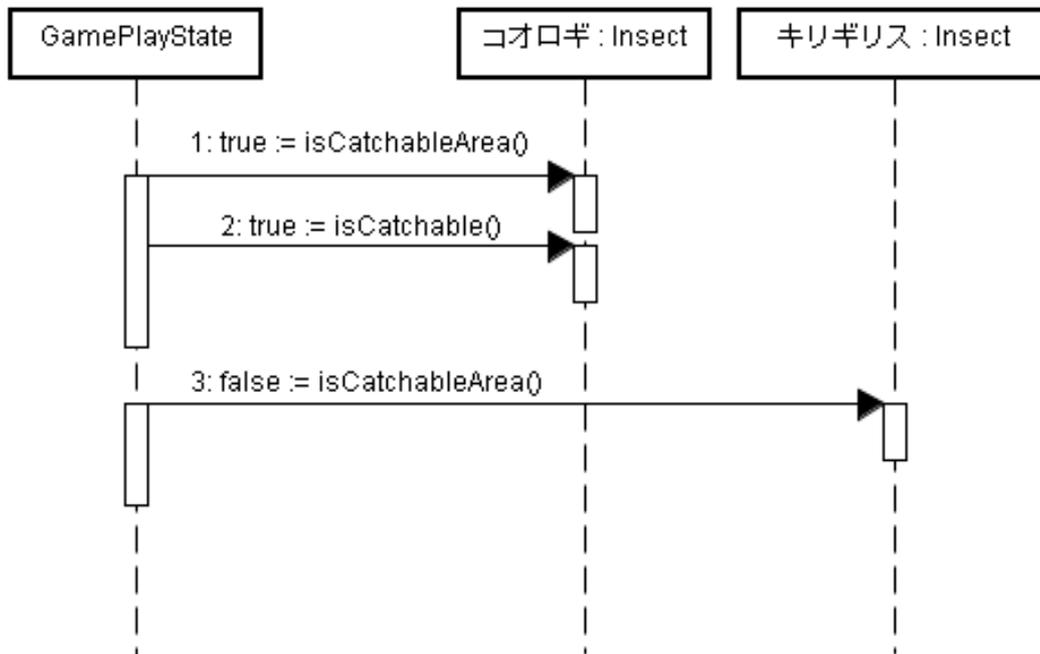



図 3-1：虫捕り変更前のシーケンス図

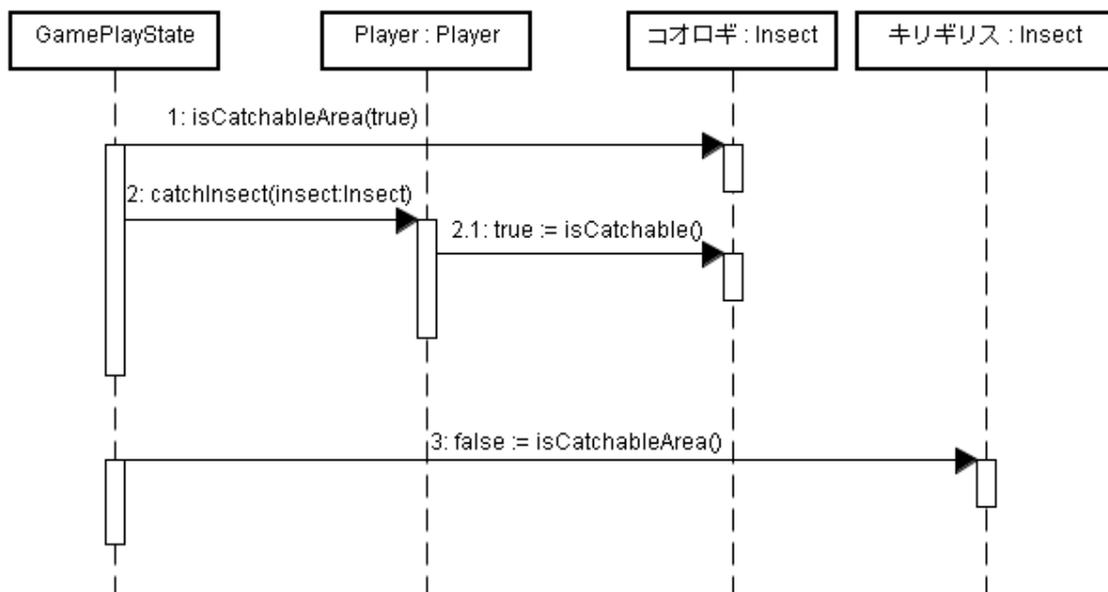



図 3-2：虫捕り変更後のシーケンス図

● 音の鳴らし方の変更

音を鳴らす際，今までは GamePlayState が SoundPlayer を持っており，BGM や虫，動物などすべての音を鳴らすようにしていた（図 3-3 参照）．しかし，GamePlayState の責任が大きくなってしまい，また GamePlayState 自体も肥大化していたため，虫や動物については自分で SoundPlayer を持ち，オブジェクトごとに音を鳴らすように変更した．

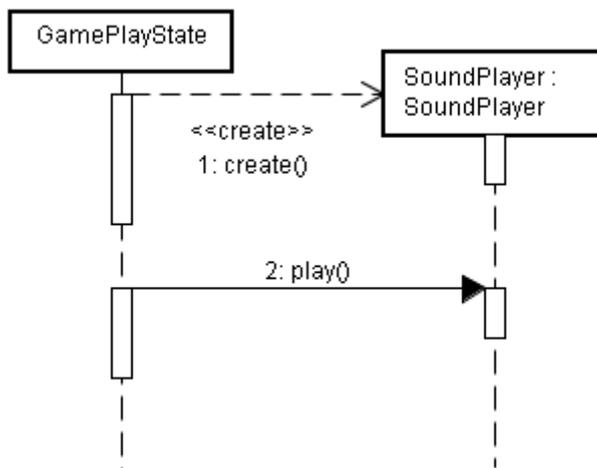



図 3-3：音関係リファクタリング前のシーケンス図

まず，GamePlayState で音が鳴るオブジェクトを GamePlayStateSoundFactory で作成する．このオブジェクトはすべて SoundMaterial のサブクラスであり，SoundPlayer を所持している．GamePlayState は逐次，自分の持つ SoundMaterial のリストに対して音を鳴らす play メソッドを呼び，必要に応じて音を鳴らすようにした．音関係のりファクタリングを行った後のクラス図，シーケンス図はそれぞれ以下の通りである（図 3-4, 3-5 を参照）．

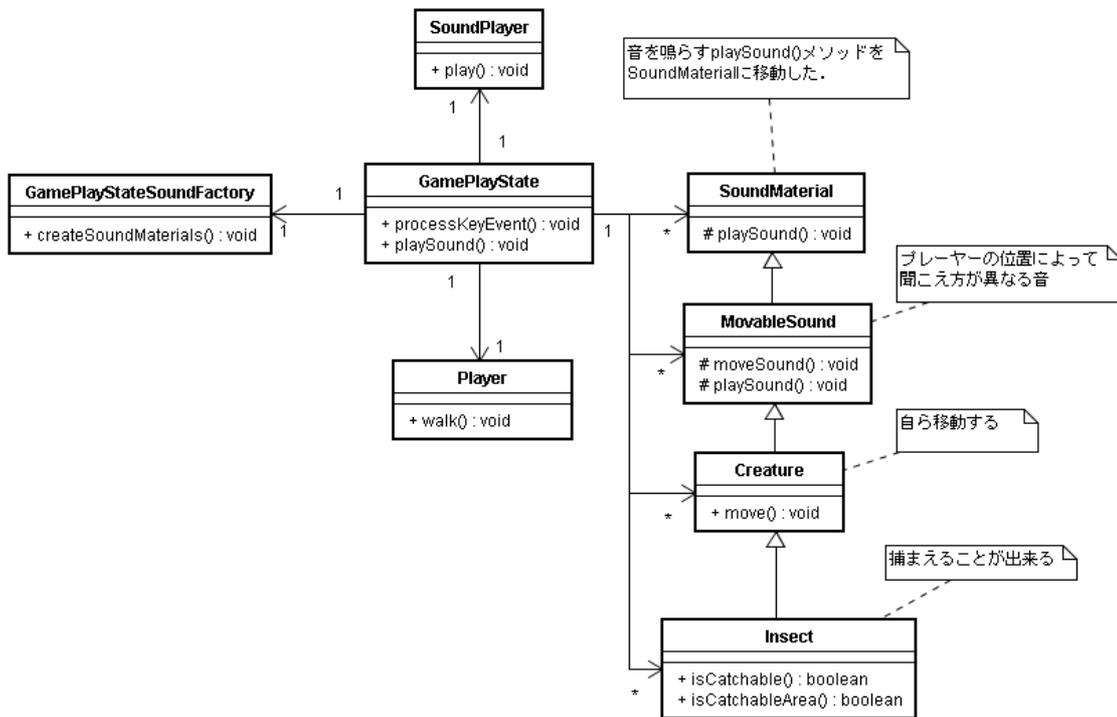



図 3-4：音関係リファクタリング後のクラス図

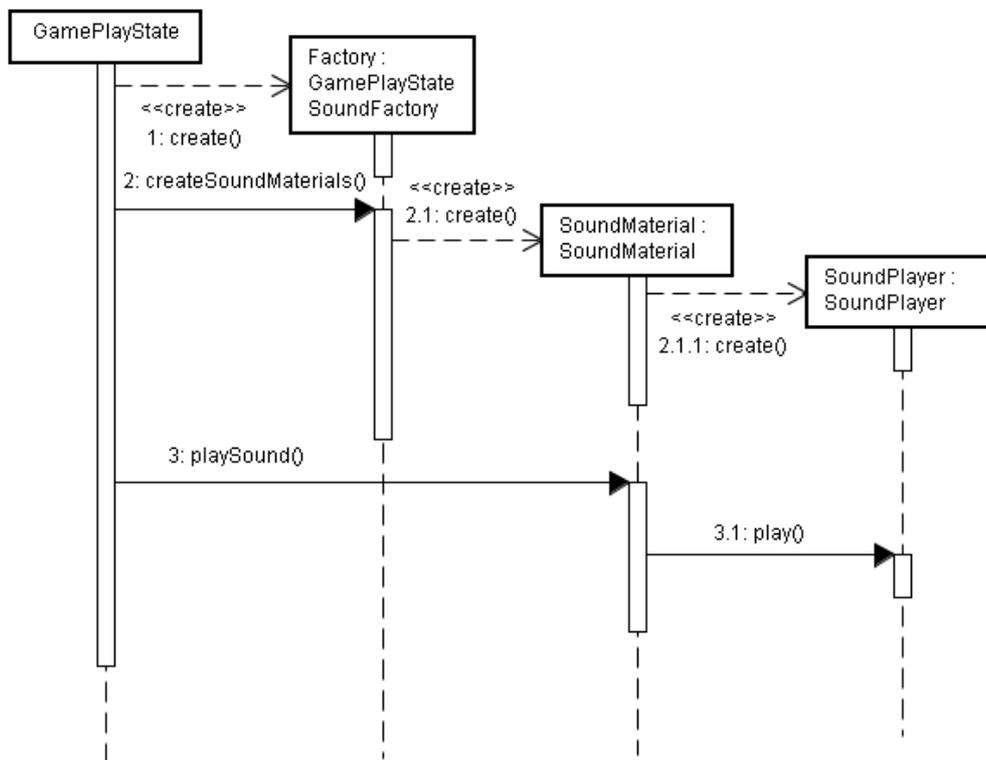



図 3-5：音関係リファクタリング後のシーケンス図

● 音ファイルの読み込みにおけるバグの修正

これまでの実装で多数の音を利用するようになったが，1）csb ファイルの容量が大きすぎると音が再生されなくなる，2）複数の csb ファイルを読み込もうとすると実行時にエラーが出る，という問題が生じた．これについては，直接 CRI・ミドルウェア社に質問し，回答をもらうことができた．

前者の問題はヒープを確保する際のバッファサイズが初期設定（3MB 程度）のままだったのが原因だった．実際にゲームで利用していた ForestWalking.csb は 7MB を超えていたので，このバッファサイズを増やすことで音を再生することに成功した．

また，後者の問題については CriAuObj をラップした関数である SoundContainer の呼び出し方に問題があることが分かった．そのため，一度現在のライブラリの利用状況とラップしている関数の関係性を整理した．

虫捕りゲームでは，音の再生に CRI Audio ライブラリを利用している．まず， CRI AudioCRAFT というデータ作成ツールを利用してゲーム内で使用する wav ファイルをキューシートバイナリ（csb）という形式に変換する（図 3-6 参照）．

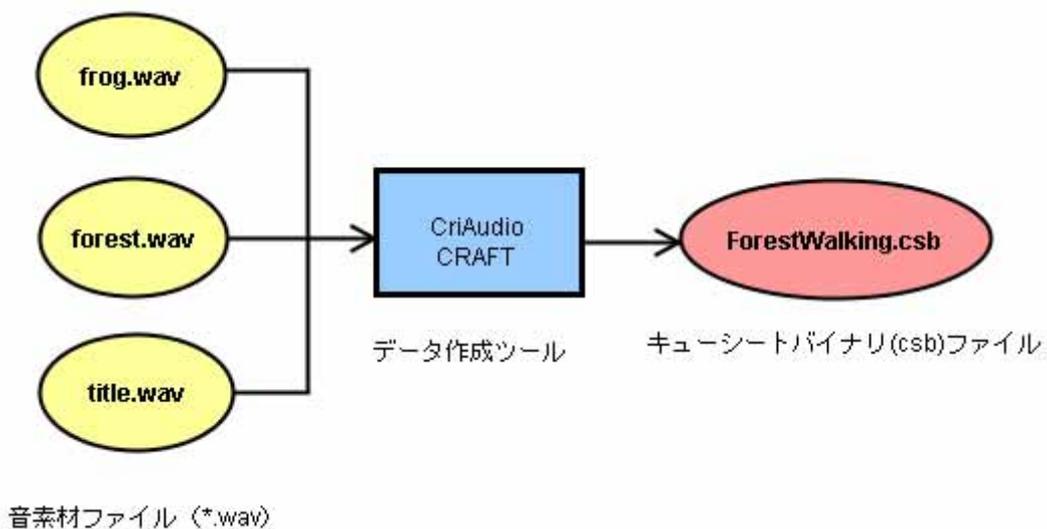



図 3-6：CRI Audio ワークフロー

プログラム内では，CriAuObject という音の再生をコントロールするオブジェクトを作成し，そこにキューシートバイナリから読み込んだデータを登録することで音の再生が可能となる．さらに，CriAuPlayer というサブクラスを利用することによって，再生中にボリュームや定位を変更するなどといった細かい制御ができるようになる．

虫捕りゲームでは，CriAuObj のほかにキューシートバイナリをロードする関数を実装してある CriAuCueSheet やサウンドを出力するための関数を実装してある CriSoundRendererBasic などをラップした SoundContainer というクラスを用意してあ

る．また，SoundPlayer が，文字列と CriAuPlayer をマッピングした連想配列である SoundShelf を所持している．この SoundPlayer が初期化される際，引数として SoundContainer を受け取り，その中でキューシートバイナリがロードされて，再生や停止が行えるようになるのである．なお，連想配列に読み込む音の役割と名前は外部ファイルに記述してある．虫捕りゲーム（ForestWalking）と CRI Audio ライブラリの利用関係を図 3-7 に図示した．

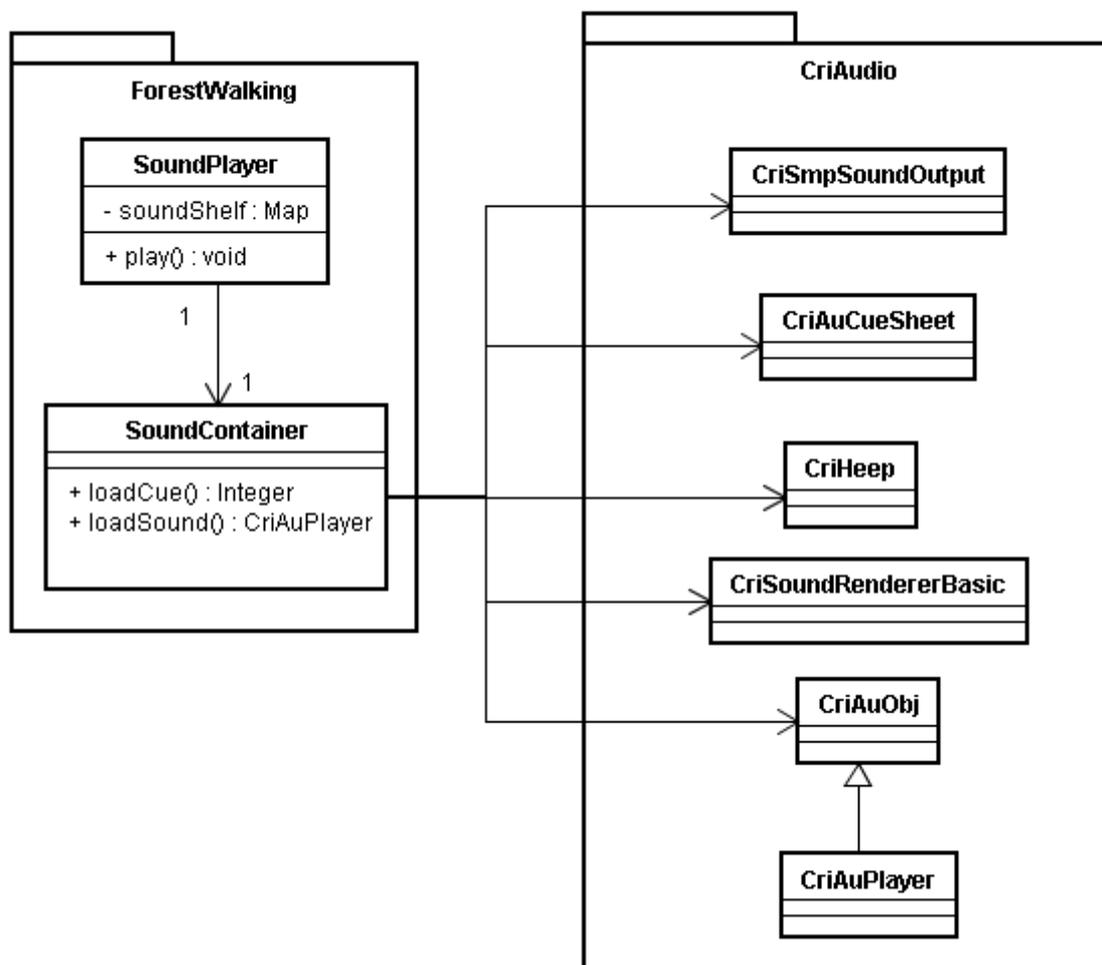



図 3-7：音関連ライブラリとの関係を表すクラス図

また，音の登録から再生・停止の手順を具体的に説明すると以下のようになる．

1. csb ファイルを作成する

CRI AudioCRAFT を使用して，キューシートとして frog.wav が登録されている Sound.csb を作成する．

2. 音の役割とファイル名を外部ファイルに記述する
ゲーム中での役割を表す名前が sound_of_frog（蛙の鳴き声）とするため，外部ファイル（sound.txt とする）に，【sound_of_frog frog】と半角スペース区切りで記述する．

3. csb ファイルのパスを引数にして，SoundContainer を作成する

```
soundContainer = new SoundContainer("Sound.csb");
```

SoundContainer の初期化の際, csb ファイルのロードやキューシートの登録が行われる．

4. SoundContainer と外部ファイルのパスを引数として，SoundPlayer を作成する

```
soundPlayer = new SoundPlayer(soundContainer, "Sound.txt");
```

SoundPlayer の初期化の際，Sound.txt を解析して，SoundShelf という連想配列に格納する. ここでは，frog という名前の音ファイルの再生を管理する CriAuPlayer が作成され, sound_of_frog という文字列とマッピングされる．

5. 役割の名前を引数として，SoundPlayer の play()関数を呼び出す
ここでは，SoundPlayer->play(“sound_of_frog”)と記述する．すると，SoundShelf の中から sound_of_frog という名前にマッピングされている frog の CriAuPlayer を探す. CriAuPlayer が見つかった場合，その play()関数を呼び出すことで，frog を再生する．

複数の csb ファイルを読み込もうとすると実行時にエラーが出る, という問題については SoundContainer を作成する際に，サウンド出力を行う CriSmpSoundOutput を複数作成しているためにリソースの確保が正しく行えていないということが原因であった．そのため，複数の csb ファイルを読み込む場合はサウンド出力の初期化を 1 回行い，キューシートハンドルの作成とキューシートロードを複数行い, 1 つのオーディオオブジェクトに対して複数アタッチするという方法に変更することで問題を解決した.

● 実行速度が遅い問題の解決

既存のバグとして最後に残ったのが，実行速度が遅いという問題である．マシンスペックにある程度影響されるのだが，Pentium3（933MHz）・メモリ 512MB のパソコンで正常に動作しないのは問題があると考えた．デバッガを利用してどの関数が動作を重くする原因となっているのかを調べた．その結果，デバッグ用に文字を描画している関数が動作を重くしている原因だということが分かった．

虫捕りゲームでは，文字を描画する際に SDL（Simple DirectMedia Layer）というライブラリを利用している．これはゲームなどのマルチメディア関連のソフトウェアを開発するための，グラフィックやサウンド等の API を提供するライブラリである．このライブラリの中では文字を作るときに以下のようなプログラムを書く．

```
1 SDL_Color textColor = {255, 255, 255}; // 文字の色を設定する（ここでは白）
2 TTF_Font *font = TTF_OpenFont("Headache.ttf", 28); // フォントを設定する
3 string textString = "Test"; // 描画する文字列
4 SDL_Surface *text = TTF_RenderText_Blended(font, textString.c_str(), textColor);
  // フォントと文字列から描画する文字を作成する
5 drawText(10, 10, text, mainScreen); // 文字を描画する
6 TTF_CloseFont(font); // フォントを解放する
7 SDL_FreeSurface(text); // テキストが占有していたメモリを解放する
```

虫捕りゲームでは，この文字作成を簡易化するために createText()という関数を作成してその中で 1~4 行目の処理を行っていた．しかしこの関数では文字列の作成は行うものの，その後に行わなければならないメモリの解放を行っていなかった．これは，5 行目の文字の描画を行う前に 7 行目のメモリ解放を行ってしまうと，本来描画するべき文字が描画できなくなってしまうためである．

関数化したことで文字の作成と描画を分けてしまったため，文字を描画するたびにメモリが解放されずにどんどんメモリリークが起こるという状態になっていたのが，実行速度低下の原因であった．この問題を解決するために，createText()関数は廃止し，文字を描画するときは必ずメモリの解放を行っているかを確認することにした．

4.1.3. ゲームを面白くするためのギミックの追加

● 制限時間とスコアの追加

虫を捕まえることができても，具体的な目標やゴールがないと，飽きるのが早くなってしまうし，プレイする気も起きなくなってきてしまう．そこで，ゲームに制限時間とスコアを追加し，ゲームを遊ぶ人が目標をもってプレイできるようにした．

ゲーム開始時は昼から始まり，一定の時間が経過すると，昼から夜になる．夜になると登場する虫や BGM が変化する．夜になってから，さらに一定時間が経過すると，ゲーム終了となり，最後に捕まえた虫の数と，スコアが読み上げられる．ゲームを遊ぶ人は，より多くの虫を捕まえ，高いスコアを獲得することを目標にプレイすることで，よりゲームに対する熱中度が向上する．

まず，時間帯の変更という要素を追加するために，今までの設計を拡張した（図 3-8 参照）．具体的には，GamePlayState の親クラスとして State クラスを，子クラスとして AftenoonState と NightState を追加した．AftenoonState は昼に登場する音を管理し，NightState は夜に登場する音を管理する．GamePlayState では，昼夜問わず登場する音を管理する．また，それぞれの時間帯が持つ音を作成するクラスとして SoundFactory を作成した．

時間帯の切り替えを明確にするために，昼から夜に切り替わる際には昼のみに登場する動物や昆虫の鳴き声をフェードアウトさせた．さらに，プレーヤーの声と鐘の音，カラスの鳴き声などで夕方ということを演出した．また，夜にはすべての音をフェードアウトさせ，プレーヤーの声でゲーム終了が近づいていることを喚起することで，ゲームが終了することを分かりやすくした．昼と夜の切り替えの方法については次項の「タイトルやチュートリアルを追加する」で説明する．

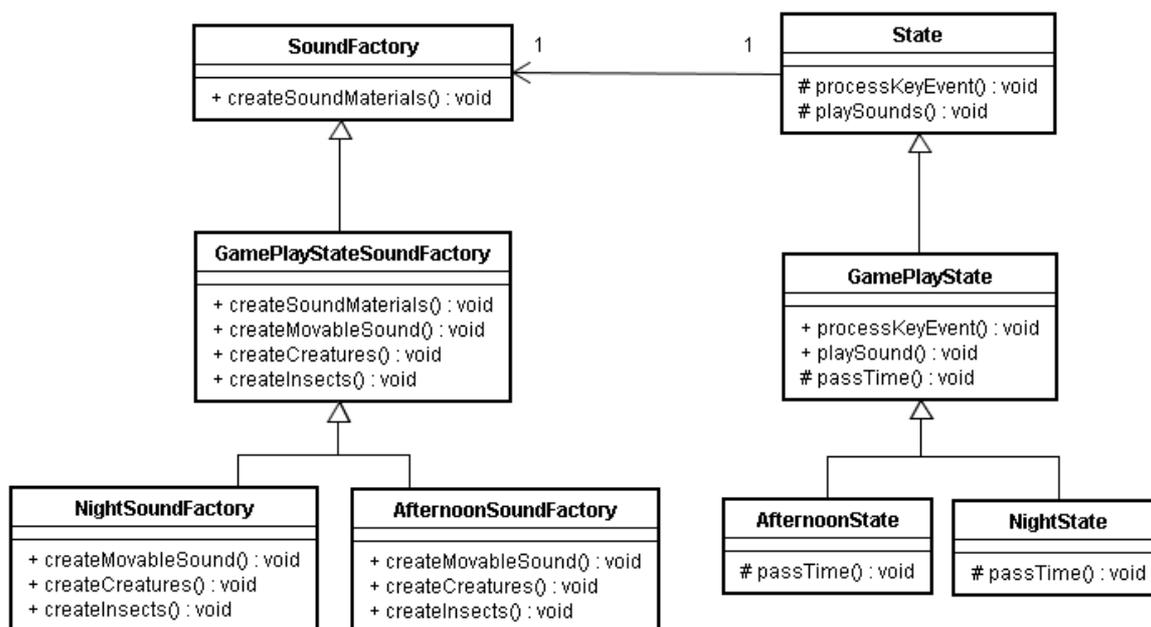



図 3-8：時間帯要素を追加したクラス図

次にスコア機能を実装するために，ResultState を追加した．捕らえた虫を虫かごに入れておくことで，ResultState でその虫かごに入っている虫の名前と数を読み上げるようにした．また，虫ごとにスコアを設定することで，合計スコアを発表するようにした．スコアの読み上げについては，現在は 0～10 までの音声を組み合わせて対応しているが，イントネーションや各桁のつながりが不自然である．これについては，各桁の数字を読み上げた音声を用意すれば解決できそうであるが，これはまだ未実装である．

（例）765 という数字を読むとき
現状：1～10 までしか音声がないので「なな，ろく，ご」と読む
改善後：１～9，10～90（10 ずつ），100～900（100 ずつ）の計 30 種類の音声を用意することで，「ななひゃく，ろくじゅう，ご」と読むことができる

● タイトルやチュートリアルなどを追加する

従来の虫捕りゲームは，説明や練習が無く突然ゲームが始まってしまうため，操作方法がわからなかったり，虫の鳴き声が聞き分けられなかったりすることが多い．そのため，ゲームの操作方法を把握しゲームに慣れてもらうために，ゲームの解説や練習モードを追加した．ゲームの解説では，ゲームの目的や操作方法や登場する虫の解説等をする．練習モードは，虫を一匹だけ登場させその虫を捕まえることでゲームの雰囲気を知ってもらう初心者のためのモードである．解説と練習モードがあることで，ゲームを遊ぶ人は操作方法を理解することができる．またゲームに慣れることができ，スムーズにゲームの本編に入ることができるようになる．

前項の制限時間やスコアを追加する際にゲームの状態を追加したので，一度状態遷移図を書くことで全体を整理した（図 3-9 参照）．

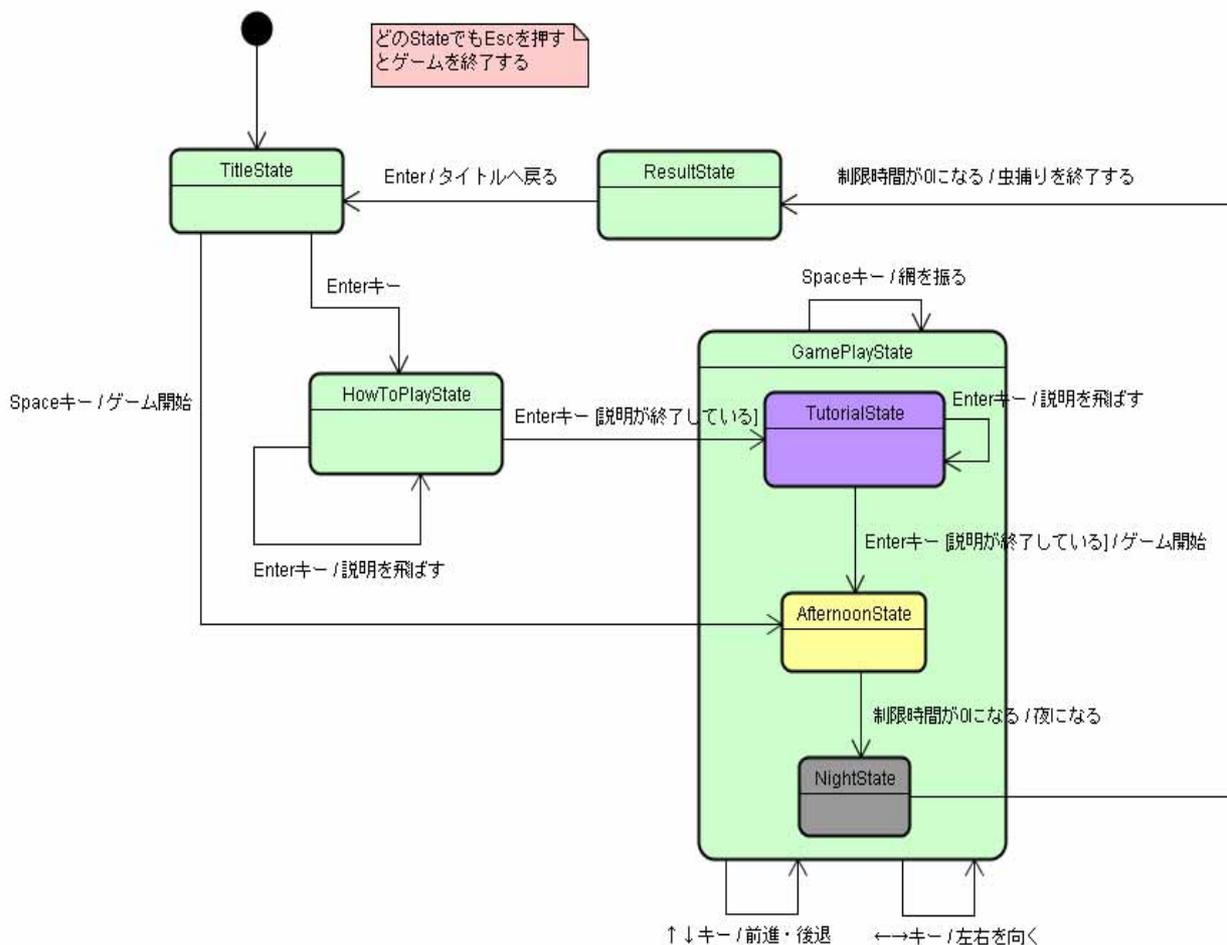



図 3-9：状態遷移図（ゲームの流れ）

この状態遷移図に従い，状態を遷移させるために State パターンを適用した．これは

State が増えることによって if 文が増えることを避けるためである．例えばある状態から別の状態へ遷移するときは以下のようなコードを書く．

```
changeState(new AfternoonState(soundContainer, mainScreen)); // 昼に遷移する
```

changeState()はすべての状態の親クラスである State クラスが持っている関数で，引数に遷移先の状態のインスタンスを受け取る．ここでは，AfternoonState に遷移させたいので AfternoonState クラスのインスタンスを作って渡している．State クラスを拡張したクラス図については図 3-10 を参照してほしい．

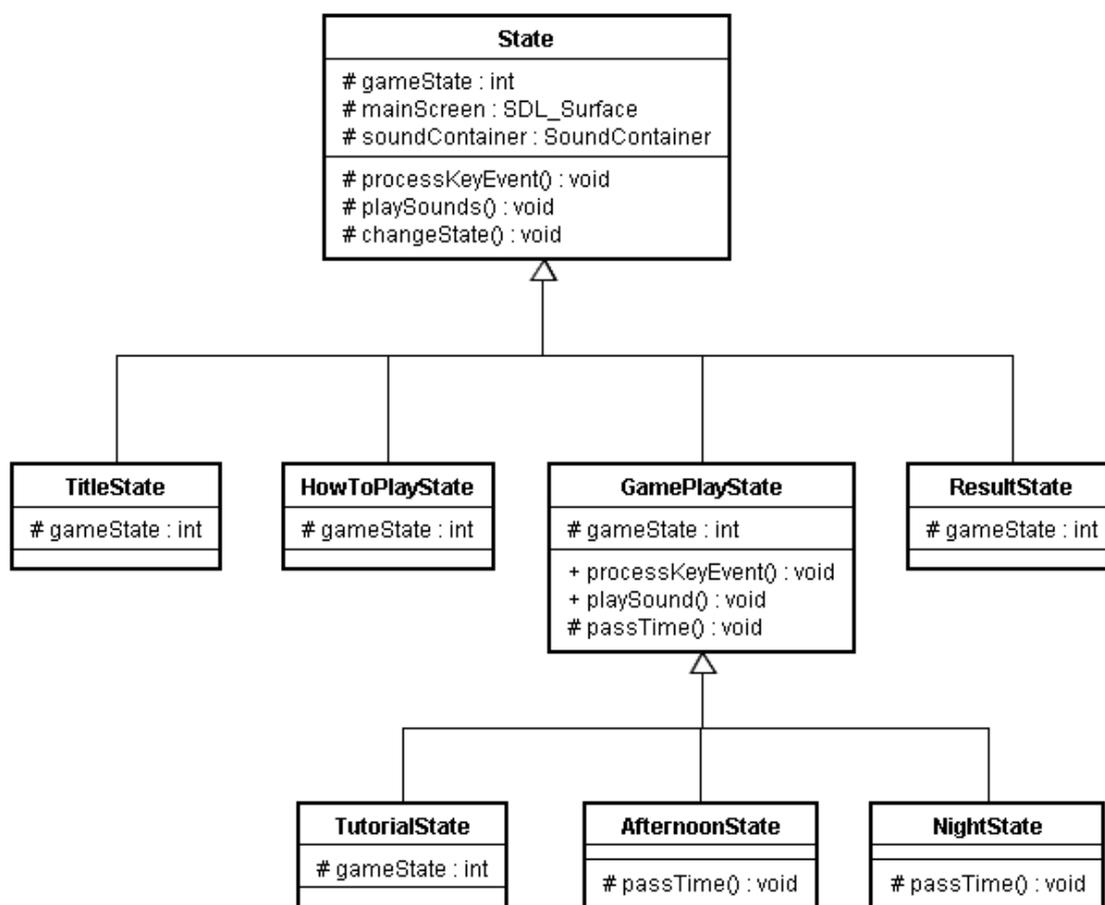



図 3-10：State パターン適用後のクラス図

● 虫が簡単に捕まえられないようにする

今までは，ある程度虫に近づいて網を振るだけで確実に虫が捕まえられた．しかし，あまり簡単に虫を捕まえることができてしまうと，張り合いがなくなり，ゲームとしても単調になってしまい，すぐに飽きてしまう．そのため，簡単には虫を捕まえることができないようにするために，虫を捕まえようとしても逃げるようにした．

プレーヤーが虫を捕まえることができる範囲にいたとしても，虫から遠ければ捕まえようとしても虫が逃げる可能性が高くなる．逆に，虫の近くに居るときほど，虫を捕まえようとしたときに，捕まえられる可能性が高くなる．このとき虫を捕まえることが出来る確率の計算には以下の式を使っている．

```
int probability = 100 - distance; // 距離から確率を計算する
```

ここでは，distance はプレーヤーと虫との Y 座標の差である．100 以上ドット離れていれば probability の値は 0 となり，近づくにつれて propbability の値が大きくなっていく．probability が 0～100 の範囲にない場合は 0 とする．こうして計算された probability を 0～100 までの間で発生させた乱数と比較することで，虫を捕まえられるかどうかを計算している．また，虫を捕まえるのに失敗したときは，虫はプレーヤーの左，左上，上，右上，右のいずれかの方向にランダムで逃げる．

- プレーヤーが移動できる範囲を限定する

今までは，プレーヤーは際限なくマップ上を移動できてしまい，あまり遠くまで行ってしまうと全く虫や川などの音が聞こえなくなってしまい，迷子になってしまうことがあった．そこで，プレーヤーが移動できる範囲を制限し，虫や川などの音が聞こえる範囲内のみ移動できるようにした．

移動できる範囲は，Config.txt で指定できる．Config.txt の ariaSize に 800～3000 の間の数値を記述することで，その範囲が移動できる範囲となる．この範囲を超えて移動しようとすると，虎の鳴き声とアナウンスが入り，プレーヤーが先に進めないようになっている．

また，マップ上のオブジェクトも，移動できる範囲外に生成されることが内容にしているが，その際にある問題点が発生した．プレーヤーが移動する際には，プレーヤー自体を動かすのが一般的であろう（図 3-11 参照）．

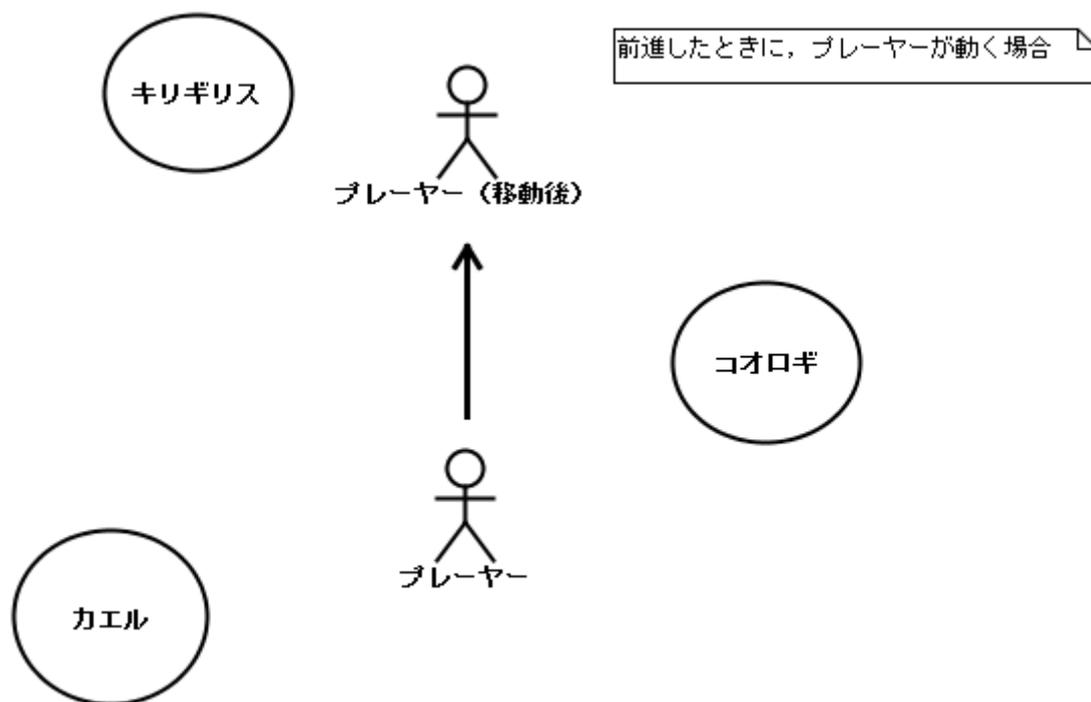



図 3-11：前進する際にプレーヤーが動く場合のイメージ

しかし，虫捕りゲームでは，プレーヤー自体は移動させず，プレーヤー以外の全てのオブジェクトを動かすことで，プレーヤーが動いているように見せている（図 3-12 参照）.

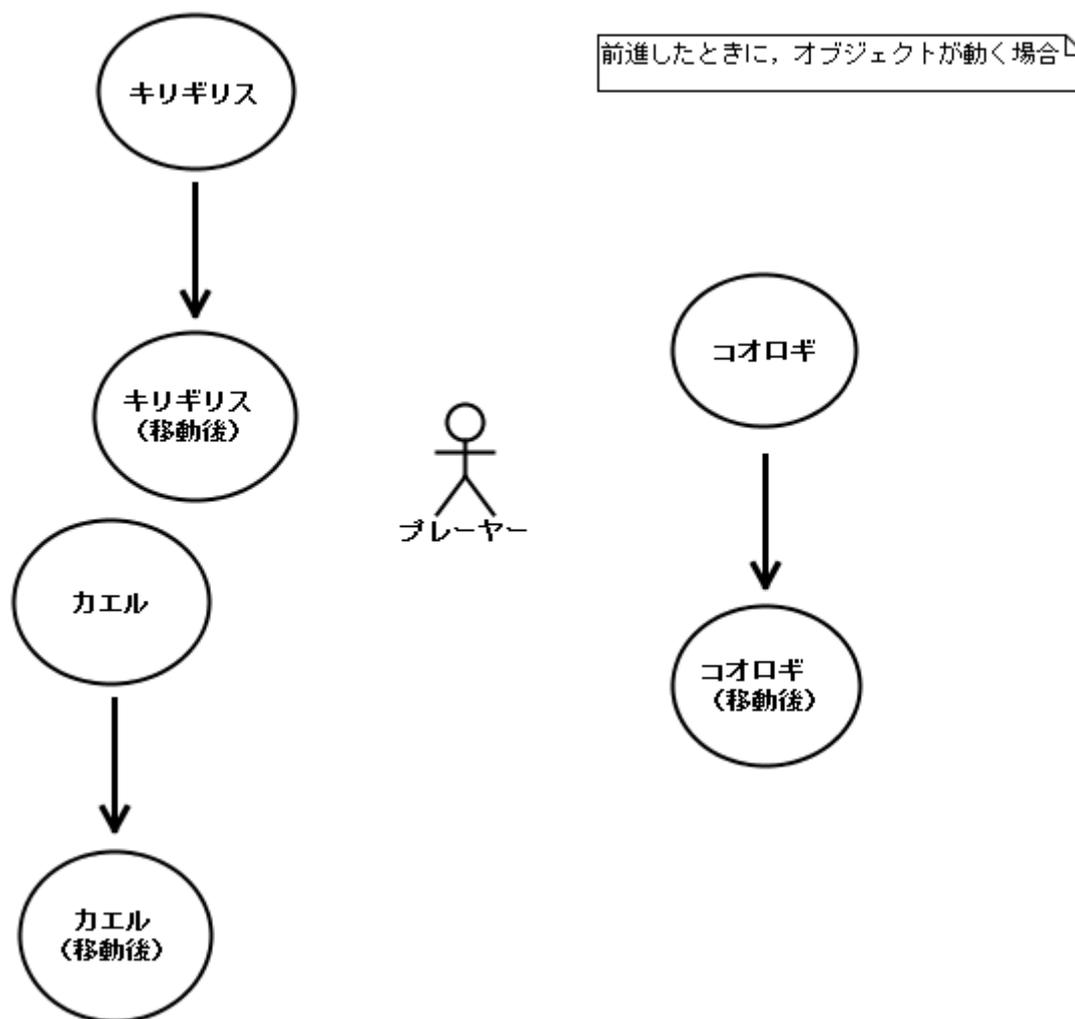



図 3-12：前進した際に虫が動く場合のイメージ

そのようにした理由は，プレーヤー自体が動いてしまうと，プレーヤーから見たオブジェクトの位置関係や角度等の特定が難しくなってしまい，位置関係に応じて音の音量や定位を調節するのが難しくなってしまうためである．

しかし，虫が生成される時の虫の座標は，画面の中心の座標を基準にして決定されるようになっていた．そのため，プレーヤーが昼の間に移動可能範囲ぎりぎりのところまで移動していた場合，夜に移り変わるときに，プレーヤーの位置ではなく，現在の画面の中心を基準に夜の虫の座標が決まってしまうので，プレーヤーが移動できる範囲を超えて虫が生成されてしまっていた．この問題を解決するために，夜に虫が生成される際には，プレーヤーの移動距離を減算して虫の座標を決定するように改良した．そうすることにより，プレーヤーが移動可能範囲ぎりぎりに移動していた場合でも，移動範囲を超えて虫が生成されてしまうという問題は発生しなくなった．

● 音の充実化

ゲームに使われている音や音声の種類を増やしたり，現状の音を洗練させたりすることで，より臨場感あふれる音環境を作成することができると考えた．そのために，現在ゲーム内で使われている音を表にまとめて，音の名前と目的などを記述した．またセリフの場合はその内容も併せて記述した（表 3-1 参照）．

この表を作ることで，後で他の人に音だけを発注するといったことができるようになったと同時に，ゲーム内で使われているすべての音と役割が一目瞭然となったので，利用できる音ファイルの名前を探すといった時間の浪費を節約することができた．

表 3-1：ゲーム中で使用する音のリスト

No.	カテゴリ	ファイル名	説明
1	背景音	forest1	森
2		forest2	森
3		forest3	森
4		forest4	森
5		forest5	森
6		forest6	森
7		night	夜の森
8		bell	寺の鐘
9		crow	カラス (昼→夜を表す)
10			
11	効果音	river	川
12		wind	風が吹く
13			
14	プレーヤーの動作	step_leaves_left	左足で歩く
15		step_leaves_right	右足で歩く
16		step_water_left	左足で歩く (水の中)
17		step_water_right	右足で歩く (水の中)
18		swing	網を振る
19			
20	プレーヤーの声	yell1	網を振るとき
21		yell2	網を振るとき
22		catch_frog	蛙を捕まえたとき
23		catch_grasshopper	キリギリスを捕まえたとき
24		catch_cricket	コオロギを捕まえたとき
25		catch_mole_cricket	オケラを捕まえたとき

26		fail_in_catch1	失敗したとき
27		fail_in_catch2	失敗したとき
28		fail_in_catch3	失敗したとき
29		cannot_go_forward	移動可能エリアの限界に来たとき
30		find_net	虫捕り網を見つけたとき
31		start_collect_insects	虫捕りを始めるときのセリフ
32		get_dark	昼→夜になったときのセリフ
33		walk_home	虫捕りを終わるときのセリフ
34			
48	TitleState 関連	title	ゲームタイトル
49		how_to_start	ゲームの始め方
50			
51	HowToPlayState 関連	explanation_of_skip	説明をスキップできることを知らせる
52		explaination_of_goal	目的の説明
53		how_to_play	操作説明
54		explaination_of_insects	昆虫の鳴き声の説明
55		sound_of_frog	カエルの声の説明
56		sound_of_grasshopper	バッタの声の説明
57		sound_of_cricket	コオロギの声の説明
58		sound_of_mole_cricket	オケラの声の説明
59		go_tutorial	チュートリアルへ行くことを知らせる
60			
61	TutorialState 関連	tutorial	移動のチュートリアル
62		end_of_tutorial	ゲームを開始することを知らせる
63			

64	ResultState 関連	go_title	タイトルへ行くことを知らせる
65		0	数字を読み上げる音
66		1	数字を読み上げる音
67		2	数字を読み上げる音
68		3	数字を読み上げる音
69		4	数字を読み上げる音
70		5	数字を読み上げる音
71		6	数字を読み上げる音
72		7	数字を読み上げる音
73		8	数字を読み上げる音
74		9	数字を読み上げる音
75		10	数字を読み上げる音
76		20	数字を読み上げる音
77		30	数字を読み上げる音
78		40	数字を読み上げる音
79		50	数字を読み上げる音
80		60	数字を読み上げる音
81		70	数字を読み上げる音
82		80	数字を読み上げる音
83		90	数字を読み上げる音
84		100	数字を読み上げる音
85		200	数字を読み上げる音
86		300	数字を読み上げる音
87		400	数字を読み上げる音
88		500	数字を読み上げる音
89		600	数字を読み上げる音
90		700	数字を読み上げる音
91		800	数字を読み上げる音
92		900	数字を読み上げる音
93		result	結果を知らせる音声
94		pre_tag_of_score	結果を知らせる音声
95		post_tag_of_score	結果を知らせる音声
96		number_of_frogs	蛙の数を知らせる音声
97		number_of_grasshoppers	キリギリスの数を知らせる音声
98		number_of_crickets	コオロギの数を知らせる音声
99		number_of_mole_crickets	オケラの数を知らせる音声

表 3-1：ゲーム中で使用する音のリスト（続き）

No.	備考
1	
2	現在未使用
3	現在未使用
4	現在未使用
5	現在未使用
6	現在未使用
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	「えいっ」
21	「えいっ」
22	「蛙をつかまえた」
23	「キリギリスをつかまえた」
24	「コオロギをつかまえた」
25	「オケラをつかまえた」
26	「逃げられた」
27	「あ゛ー」
28	「もう少し近づかなくちゃ」
29	「これ以上進むのは危険そうだ，引き返そう」
30	虫捕り網を見つけるというギミックを実装した後で
31	「さあ，がんばってたくさん捕まえるぞ！」
32	「大分暗くなってきたな」

33	「夜も更けてきたし，そろそろ帰ろう」
34	
48	ゲーム名「ForestWalking」を入れる
	「ゲームの説明を聞く場合は，enter キーを押してください．
49	説明を聞かずにゲームを始める場合は，space キーを押してください」
50	
51	「ゲームの説明を開始します．説明をスキップしたい方は enter を押してください」
	「初めにゲームの目的を説明します．あなたは昼間の森の中を歩いています．
	聞こえてくる様々な音から虫の鳴き声を探し出して，虫を捕まえてください．
	夜になるまでにできるだけたくさんの虫を捕まえましょう．
52	捕まえた虫の種類と数に応じて得点が計算されますので，高得点を目指してください．」
	「次に操作方法を説明します．上カーソルは前進，下カーソルは後退です．
	左右のカーソルでは自分の向いている向きを変えることができます．
	Space キーを押すと虫捕り網を振ります．虫を捕まえるためには上下左右の
	カーソルを使って出来るだけ虫に近づいて網を振ってください．
53	また，Esc キーを押すとゲームを終了します．」
	「続いて，捕まえられる虫について説明します．
	捕まえられるのはキリギリス，コオロギ，オケラ，カエルです．
54	それぞれの鳴き声を聞いてみましょう．」
55	「これはカエルの鳴き声です」
56	「これはキリギリスの鳴き声です」
57	「これはコオロギの鳴き声です」
58	「これはオケラの鳴き声です」
	「最後に，ゲームを始める前に少し練習してみましょう．
59	練習を始めるには Enter を押してください．」
60	
	「虫の鳴き声を頼りに上下左右のカーソルを使って移動しましょう．
	虫が近くにいると思ったら space キーで網を振ってください．
	ただし，虫に十分近づいていないと虫が逃げてしまうので注意してください．
61	それでは練習してみましょう．」
	「虫の捕まえ方は分かりましたか？　以上で練習は終わりです．
	実際のゲームを開始するには enter キーを押してください．
62	もう一度説明を聞きたい場合は，space キーを押してください．」
63	
64	「enter キーを押すと，タイトルに戻ります．」
65	数字は自然に聞こえるようにしたい

66	
67	
68	
69	
70	
71	
72	
73	
74	
75	
76	
77	
78	
79	
80	
81	
82	
83	
84	
85	
86	
87	
88	
89	
90	
91	
92	
93	「結果発表です」
94	「スコアは」
95	「～です」
96	「蛙を捕まえた数」
97	「キリギリスを捕まえた数」
98	「コオロギを捕まえた数」
99	「捕まえたオケラの数」となっているので，「オケラを捕まえた数」に直したい

4.1.4. デバッグ・修正およびリファクタリング２

横浜市立盲学校の生徒達や大岩研究室関係者，クライアントへのユーザレビュー（詳細は 3-2 節を参照）を通じて得られた意見をもとに，納品前に最後のデバッグ・修正およびリファクタリングを行った．またそのとき，何がタスクとして残っていてどの優先度が高いかを明確化するために，デバッグ時 TODO リスト作成した（表 3-2 参照）．

機能追加・バグ修正・リファクタリングの 3 つのカテゴリを作り，それぞれのタスクと重要度を書き込んでいった．また，そのタスクの着手状況や優先度がわかるように，未修正のもので重要度が高いものは赤，未修正のもので重要度が中程度のもの，または修正中や保留は黄，修正済みのものは青という色分けを行った．

表 3-2 デバッグ時 TODO リスト

作業カテゴリ	作業内容	状態	優先度
機能追加	練習モードを作成する	修正済み	高
	説明をつける	修正済み	高
	音の高低差をつける	未修正	低
	音の種類を増やす	修正済み	中
	虫取り網を探す	未修正	低
	森の一定範囲外に出られないようにする	未修正	高
	デバッグモードをオフにする機能をつける	修正済み	中
	スコアの読み上げを自然にする	修正済み	中
	スコアの評価をつける（がんばりました，もっとがんばりましょう等）	未修正	中
	ステージ編成を追加する	未修正	中
バグ修正	Result がおかしいバグを修正する	修正済み	高
	フェードイン･フェードアウトを修正する	未修正	中
	KeyRelease を実装する	修正済み	高
	川の音の聞こえ方を修正する	未修正	高
	一定範囲外からはみ出てしまうのを修正する	修正済み	高
	夜になると一定範囲外に虫ができてしまうのを修正する	修正済み	高
	昼，夜のコンストラクタが重い（new するときが原因？）	未修正	高
リファクタリング	外部ファイルからの読み込み（制限時間，エリアサイズなど）	修正済み	中
	マジックナンバーの整理	修正中	中
	SoundPlayer に fade 系をもたせる	保留	中
	Factory の修正	修正済み	高
	KeyEventListener で Exit は親に書く	修正済み	中
	esc を押したときの exit(0)をやめる．	保留	高
	include の整理	修正済み	中
	移動の際，Player を動かしたい	修正済み	中
	音声読み込みのテキストの書式を分かりやすくする	未修正	低
	外部テキストから，虫の数などを読み込む	未修正	中

- ResultState のバグの修正

ResultState ではスコアの読み上げなどの結果発表を担当しているが，このときに読み上げの声がおかしくなってしまうというバグがあった．これは読み上げの際に，C++標準ライブラリの list を使っていた. しかし iterator の初期化に失敗しているようで，デバッグではうまく動くが，バイナリにすると動かなくなってしまう．このエラーの原因は未だ分かっていないが, 暫定的な対処として list の内容を一度配列に読み込んで, iterator ではなく普通の for 文を使うことにした.

- Release 環境で動作しないバグについて

本来，VisualStudio で開発したプログラムを配布するときは，コンパイル時にデバッグ版ではなくリリース版を選択するべきである．しかし，リリース版としてコンパイルすると実行時に「フォントが開けない」というエラーが出てしまう．エラーの原因について Web などを調べてみたが，どうしても原因が特定できなかったため，現在配布してあるものはデバッグ版でコンパイルしてある（デバッグ版とリリース版で動作に差異などは生じていないことを確認してある）.

- ヘッダーファイルのインクルード関係の整理

C++では，あるクラスで必要となるクラスは，ヘッダーファイルをインクルードする必要があるが，ヘッダーファイルを相互にインクルードし合ってしまうと，インクルード関係が循環してしまい，コンパイルエラーとなってしまう．この問題は，相互インクルードの関係にあるヘッダーファイルそれぞれに，直接#include の記述がある場合と，あるヘッダーファイルをインクルードした別のヘッダーファイルを介して，相互インクルードになってしまう場合がある．前者に関しては，問題となる箇所が発見しやすく，対処も容易に施せるが，後者に関しては，問題箇所を特定するために複数のインクルードファイルを点検する必要があり，インクルード関係が複雑になっているほど，問題箇所の特定と対処に手間がかかる.

そこで，このような問題が起こりにくいよう，また，起きた場合にも容易に対処できるよう，インクルード関係がなるべく簡潔になるように整理した．以下が，整理する前のヘッダーファイル間のインクルード関係を示した図である（図 3-13 参照）.

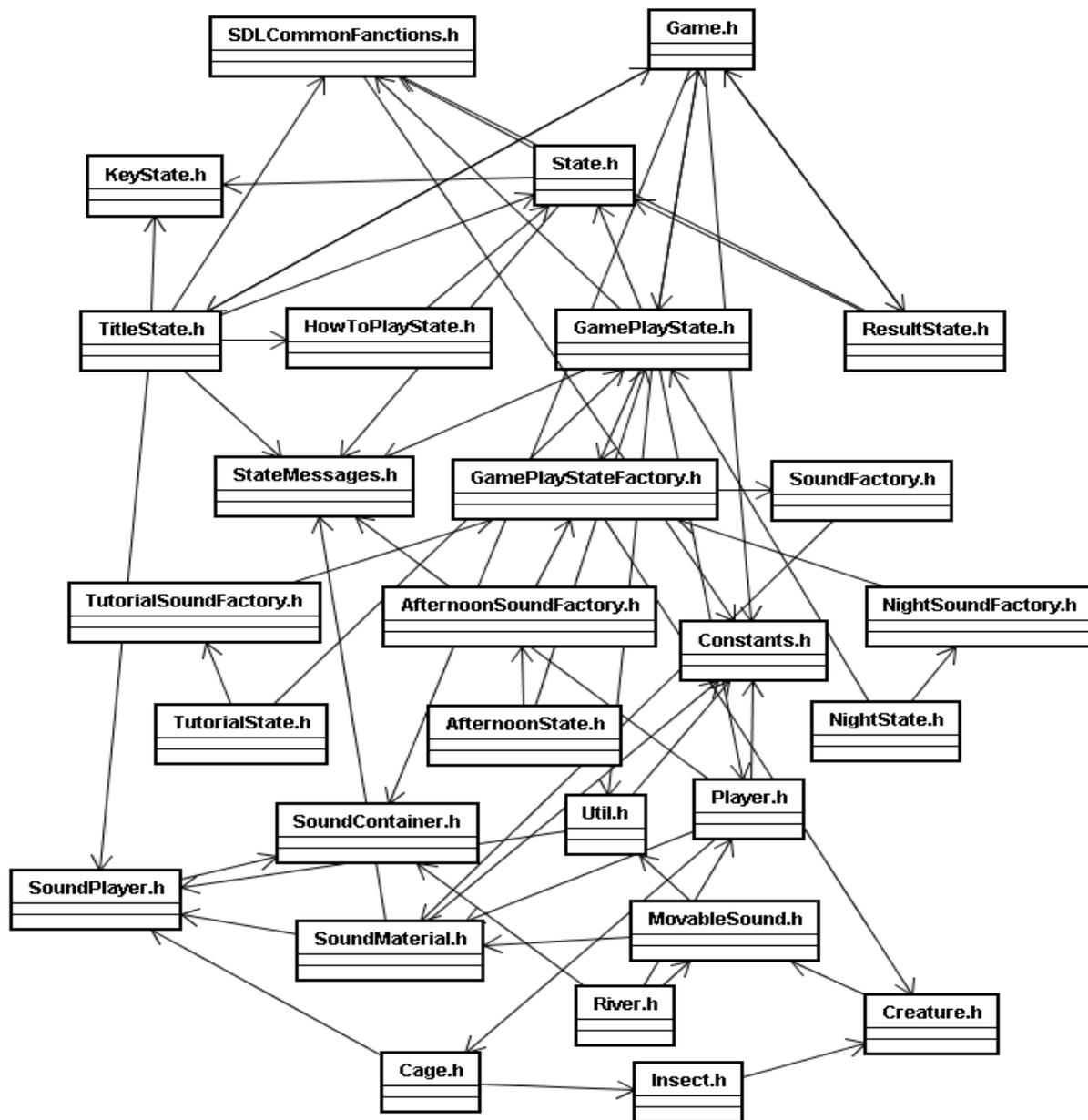



図 3-13：整頓前のインクルードファイル相関図

この図では，あるヘッダーファイルが，そこから伸びている矢印の向いている先のヘッダーファイルをインクルードしていることを示している．この図を一見してわかる通り，ヘッダーファイル間のインクルード関係が非常に複雑になっている．このような状態では，上記のような相互インクルード問題が発生した際に，問題箇所の特定と対処が非常に困難になると予想される．

そこで，ヘッダーファイル間のインクルード関係を簡潔に整理した．以下が整理後のヘッダーファイル間のインクルード関係を示した図である（図 3-14 参照）．

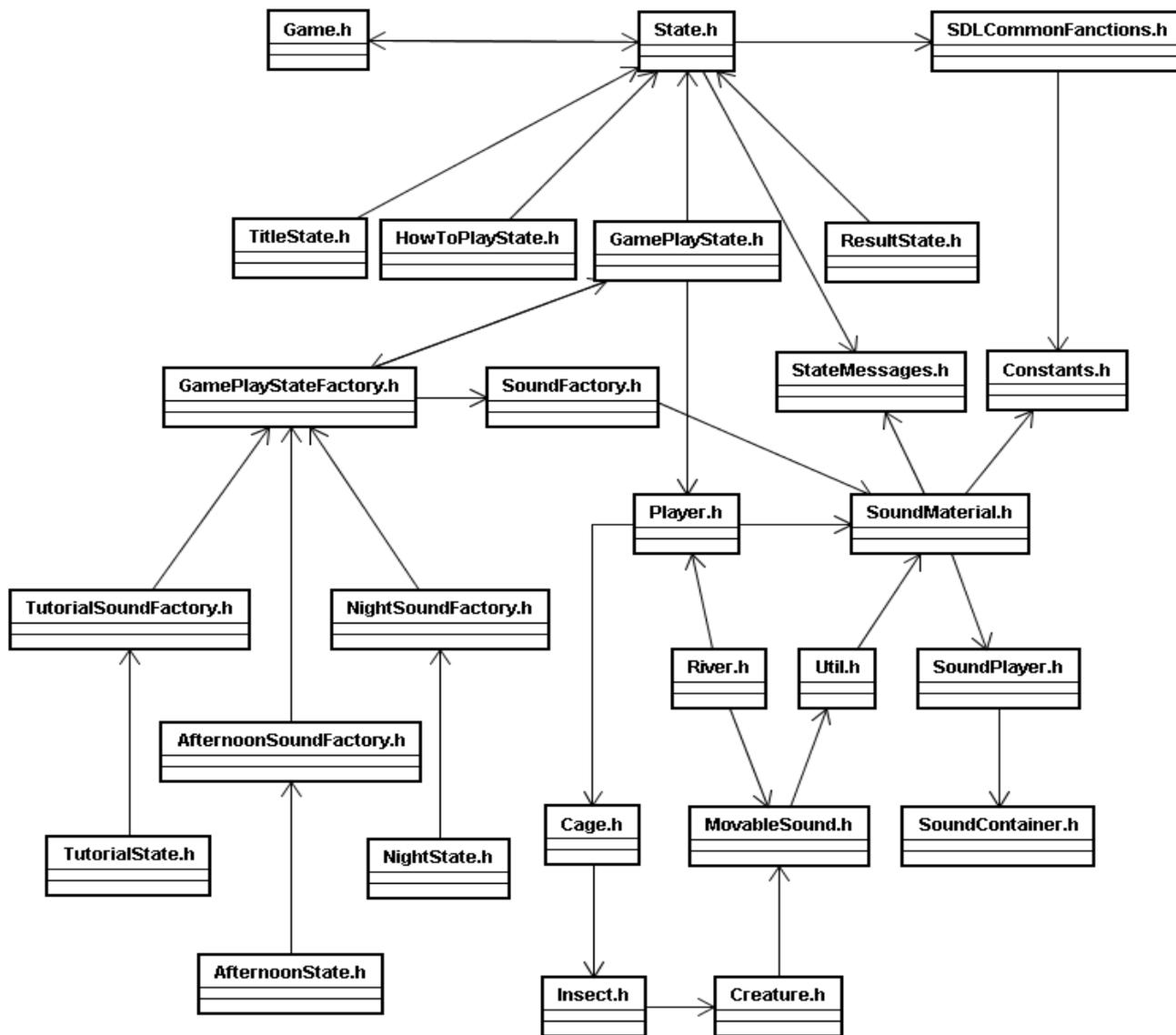



図 3-14：整頓後のインクルードファイル相関図

整理する前のインクルード関係と比べて，非常に簡潔にまとまっているのがおわかりいただけると思う．このくらい簡潔であれば，相互インクルード問題が発生した際に問題箇所の特定と対処が比較的楽に行えるだろう．また，今後新しいヘッダーファ

イルが追加されたり，ヘッダーファイル間のインクルード関係に変更が生じたりする場合にも，上記のインクルード関係を参照すれば，不要なインクルードを行うことなく，インクルード関係を簡潔に保ったまま，追加や変更ができるようになるだろう．

● 定数の管理について

従来のソースコードでは各クラスに定数が散逸していたため，ゲームの設定を変更するときに関係している定数を探しながら値を変更する必要があった．これはデバッグ時に相当な労力がかかることで，これらの定数を 1 つのファイルにまとめてしまえば，わざわざ他のファイルを開く必要がなくなると考えて，定数だけをまとめたヘッダである Constants.h を用意した．

しかし，Constants.h はその性質から様々なクラスにインクルードされるため，値を書き換えるとインクルードしているすべてのクラスをコンパイルしなければならず，結果的にコンパイルに多くの時間がかかるようになってしまった．この問題を解決するために，Constants.h の中に記述されている定数のうち特に変更回数が多い「制限時間」「マップの広さ」については，外部ファイルから読み込むことで動的に変更できるようにした．これによって，コンパイルにかかる時間が圧倒的に短縮されて，デバッグがスムーズに行えるようになった．

● CriAuPlayer の仕様と，読み上げ時の状態遷移について

CRI Audio では, CriAuPlayer に音ファイルのキューをセットして音を再生するが，そのときの CriAuPlayer の状態遷移は図 3-15 のようになっている．

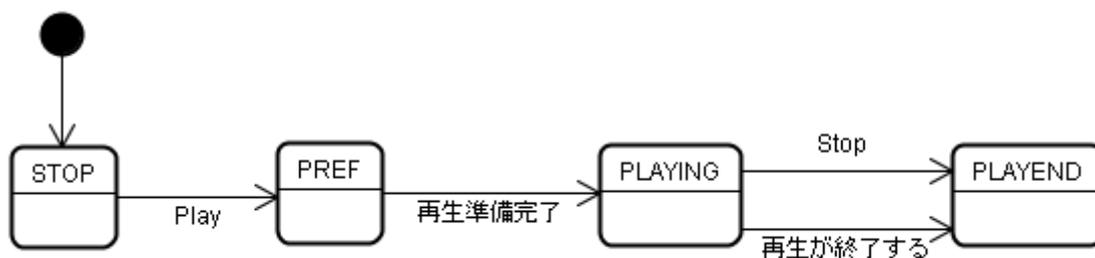



図 3-15：CriAuPlayer の状態遷移図

この状態遷移図を見ると分かるように，音の再生が終了すると CriAuPlayer は PLAYEND 状態になる．例えば，虫の数を表す時に“７”という音を再生するときは以下のような順序になる．

1. “７”がセットされた CriAuPlayer は，再生前なので STOP である
2. CriAuPlayer が STOP ならば，“７”を再生する

3. “7”の再生が終わると，CriAuPlayer は PLAYEND という状態になる

2 の処理は，ゲーム内で音を鳴らしている関数である playSounds は while ループの中で逐一呼ばれているため，1 度再生している間は同じ音は再生しないようにするために必要である．しかし，このままでは再生前と再生後の状態が異なるため，同じ音を 2 回目以降に再生しようすると，2 の条件に合致せずに音が再生されない．この状況を改善するためには CriAuPlayer が PLAYEND かどうかを調べる必要がある．上記と同じ例を用いると同じ音を 2 回以上鳴らす場合は以下のようになる．

1. “7”がセットされた CriAuPlayer は，再生後なので PLAYEND である
2. CriAuPlayer が STOP か PLAYEND ならば，“7”を再生する
3. “7”の再生が終わると，CriAuPlayer は PLAYEND という状態になる

2 回目以降でも音が再生するように，2 の条件を変えた．しかし，ここで新たな問題が発生した．この場合では，PLAYEND になった直後に 2 の条件に合致するため，同じ音が意図しなくても何度も再生されてしまうのである．このため，同じ音を 2 回以上鳴らし，さらに何度も再生しないようにするためには，CriAuPlayer の状態とは別に再生状態を持つ必要が出てくる．そこで，SoundPlayerState という状態を追加して，再生状態の管理を行うことにした．例として，ResultState の読み上げ時の状態遷移を挙げる．まず，下図を見て頂きたい．

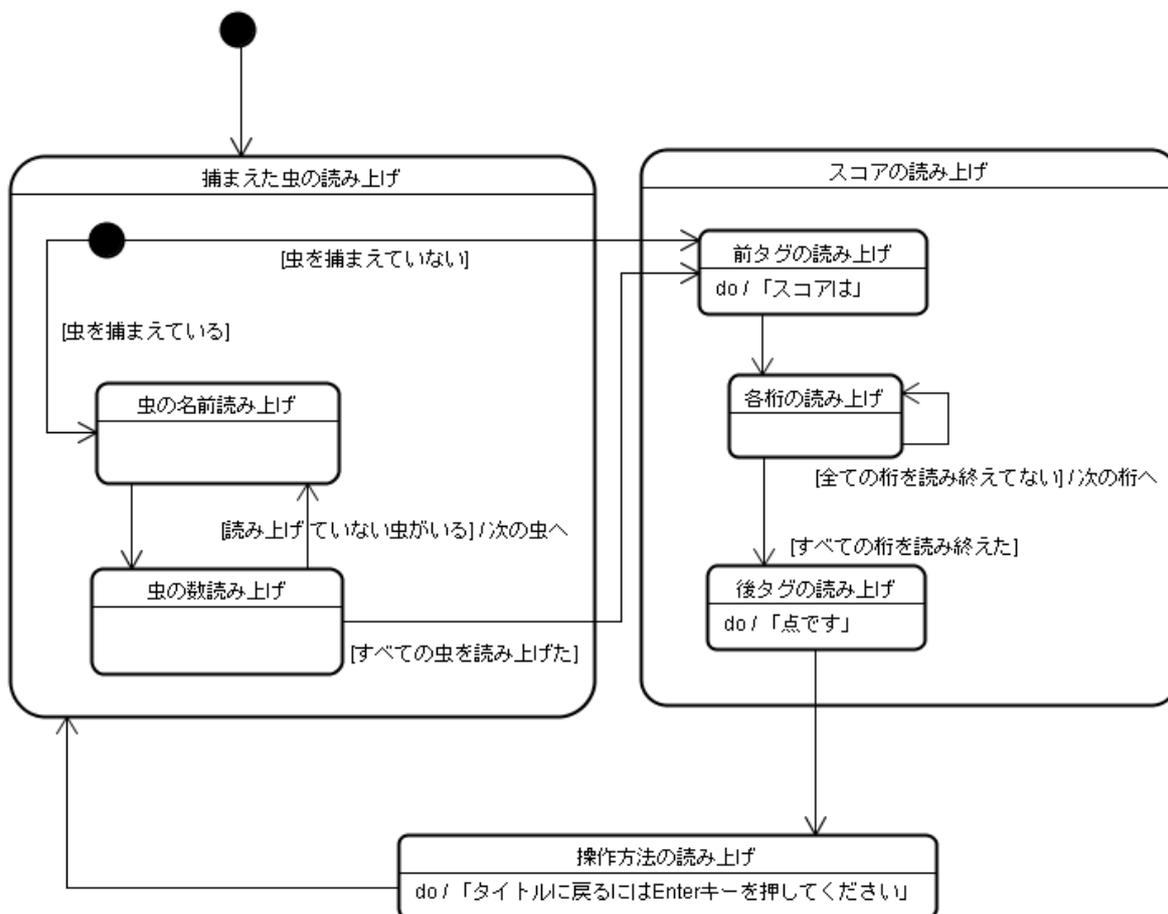



図 3-16：ResultState の状態遷移図

図3-16はResultStateにおける読み上げの状態遷移を表したものである. この中で, 捕まえた虫の名前・数を読み上げる際とスコアを読み上げる際に SoundPlayerState が必要になる. それぞれの場合における SoundPlayerState の状態遷移を表したものが図 3-17, 3-18 である.

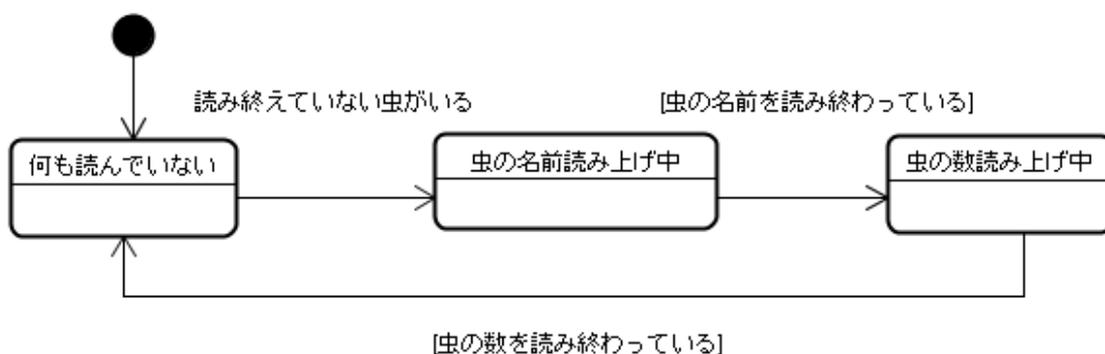



図 3-17：SoundPlayerState の状態遷移図（虫の名前読み上げ時）

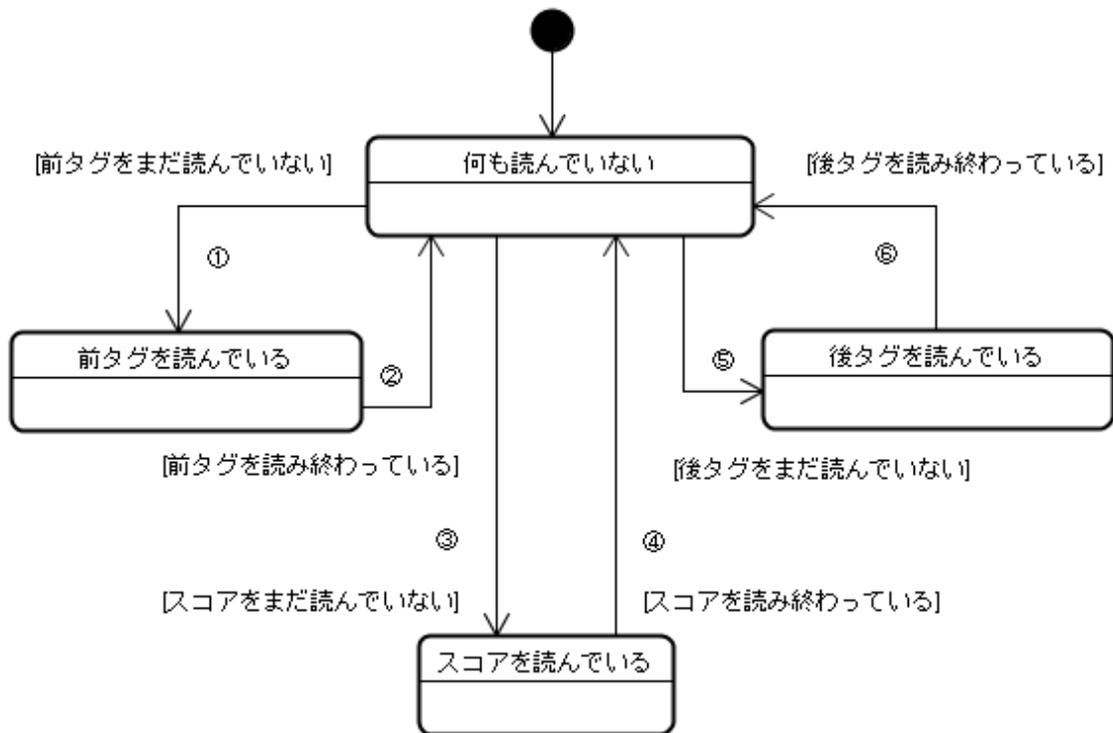



図 3-18：SoundPlayerState の状態遷移図（スコア読み上げ時）

SoundPlayerState を用いて音の再生状態を管理することで，音を再生するときの順序は以下のように改善された．もう一度同じ例を用いて説明する．

1. “７”がセットされた CriAuPlayer は，再生後なので PLAYEND である．また SoundPlayerState は「何も読んでいない状態」である．
2. CriAuPlayer が STOP か PLAYEND で，かつ SoundPlayerState が「何も読んでいない状態」ならば，“７”を再生し，SoundPlayerState を『“７”を読んでいる状態」にする
3. “７”の再生が終わると，CriAuPlayer は PLAYEND という状態になり，SoundPlayerState は「何も読んでいない状態」になる

一見，何も変わっていないようだが，例えば“７”の音が再生し終わったあとに別の音を再生したい，というような状況を考えると違いがよく分かる．SoundPlayerState を導入したことで，CriAuPlayer が PLAYEND になっても即座に同じ音がもう一度再生されるという状況を防止できる．これによって，同じ音を 2 回以上再生し，なおかつ意図しないのに何度も再生されてしまうという状況を改善できた．

● キーが過剰反応してしまうバグの修正

キーが押されているかどうかを判断するために，SDL を用いて以下のような関数を用いていた．

```
bool Util::isPressed(int id) {
    Uint8* keys = SDL_GetKeyState(NULL); // すべてのキーの状態を取得する
    return keys[id] == SDL_PRESSED; // 対象のキーが押されているかどうかを返す
}
```

しかし，この関数を使っているとキーを押しっぱなしにしたときに過剰反応してしまうというバグが見つかった．たとえば説明を読み上げているときに Enter を押すと説明を 1 つ飛ばすのだが，キーを押しっぱなしにすると Enter が何度も押されていると判断されて説明を全部飛ばしてしまうのである．押しっぱなしであると判断されない間隔がかなりシビアで，普通にキー操作しているだけでも押しっぱなしであると判断されて，ゲームの実行に支障をきたしていた．そのため，キーが押されているかを判断する関数に改良を加えて，次のような関数を作った．

```
bool Util::isPressedOnce(int id) {
    Uint8* keys = SDL_GetKeyState(NULL);
    if (keys[id] == SDL_PRESSED) {
        // キーが過剰に反応することを防ぐ
        if (!isKeyPressed) {
            isKeyPressed = true;
            return true;
        }
    } else {
        isKeyPressed = false;
    }
    return false;
}
```

まず，グローバル変数としてキーが押されたかどうかを判断する isKeyPressed という bool 型のフラグを用意して，初期値として false を与えておく．そして，キーが押された場合はこのフラグを true にする．フラグが false である最初の 1 回だけはキーが押されていると判断する．また，キーが離された時点で isKeyPressed は false に戻る．この関数を用いることで，キーの過剰反応を防ぐことができた．

4.1.5. ライブラリについて

今回使用したライブラリは，CriAudio，SDL(Simple Directmedia Layer)，Boost の 3 種類である．この節では，これらのライブラリについて解説する．

- CriAudio について

CriAudio は，サラウンド対応の音声を作成し，再生するためのミドルウェアである．C++から CriAudio のサウンドデータを再生するには, CriAudio ライブラリを使用する．

映像を用いないゲームにおいては，CriAudio は非常に利用価値が高く，有用なものであった．以下 CriAudio について解説する．

- ➢ CriAudio のワークフロー

wav 等のサウンドファイルを，CriAudioCRAFT を使用してキューシートバイナリファイルに変換し，キューシートバイナリファイルを，CriAudio ライブラリを使用して再生する（図 3.6 参照）．

- ➢ CriAudio のアーキテクチャ

CriAudio は，大きく分けて 2 つの部分から構成されている．一つは音をどのように鳴らすかを制御するボイスコントロール部，もう一つは鳴らしたい音を発生させるサウンドレンダラ部である．この二つはそれぞれ演奏者と楽器に例えることができる．

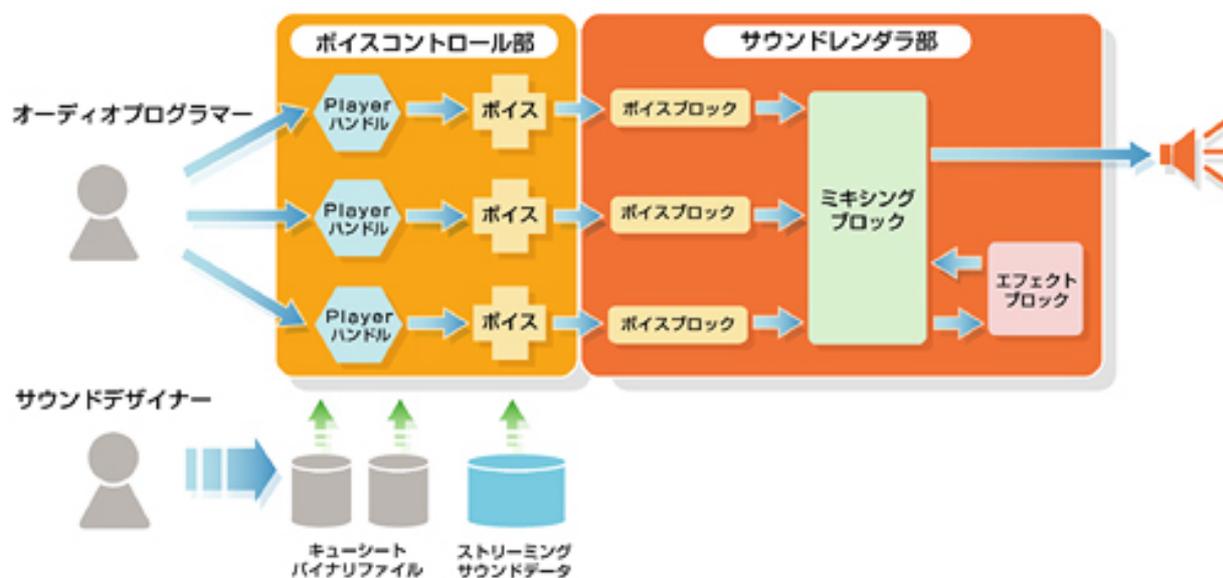



図 3-19：CriAudio のアーキテクチャ

➢ AISAC

AISAC（Advanced Interactive Sound and Active Controller）とは，回転数に応じたエンジン音や，試合の盛り上がりに呼応した競技場の歓声，発音源までの距離に応じて変化する足音や声など，ゲームの状況に応じて変化するインタラクティブなサウンドを設計するためのコンポーネントである．このコンポーネントを使うことで，虫の距離に応じて音量や定位を変化させる，音に奥行きを持たせるためにリバーブをかける等といった複雑な計算や処理を簡単に実現することができる．

➢ CriAudio を導入するメリット

CriAudio を導入するメリットは，大きく 3 つ挙げられる．

一点目は，音を作成する自由度が高いことである．定位やリバーブのコントロールはもちろん，ドップラー効果やオクルージョン（こもり音）等の音が自由に作成でき, 微調整も簡単に行える. また, 直接プログラムをいじらずに, CriAudioCRAFT 等のツールを使用することにより，GUI で簡単に音声ファイルを作成することができるので，プログラムに精通していない人が音ファイルの作成を担当することもできる．

二点目は，音の再生が容易に行えることである．作成したキューシートバイナリファイル上のサウンド名を指定するだけで再生が行える．そのため，非常に高度に作り込まれた音ファイルであっても，それを利用する側は何も意識せずにその音ファイルを利用することができる．虫捕りゲームを作成する際は，実際に音声ファイルを作る人とプログラムを作成する人を分けたが，特に不自由なく音声ファイルを扱うことができた．

三点目は，マルチコア CPU に最適化してくれることである．今回はマルチコア CPU を用いなかったが，マルチコア CPU を用いた場合，ゲームのメイン処理を行うコアとは別のコアで CriAudio の処理を行うことができるので，メイン処理に負担をかけずに音声の再生をすることができる．

➢ その他の機能

CriAudio には，ポリフォニック再生機能，ランダム再生機能，シーケンシャル再生機能，4ch サラウンドリバーブ機能，クロスフェーディングやボイスハイライトを実現するシネマティックサウンド機能，音声再生中のファイル読み込み機能といった様々な機能が搭載されている．今後，虫捕りゲームをバージョンアップさせる際に，より高度な音声処理を行う必要があるかもしれないが，そのような場合でも，CriAudio の各種機能を使用して様々な音声を作成することが可能である．

➢ 参考 URL

CriAudio に関する情報は，以下のサイトから取得することができる．

http://criware.jp/products/product_criaudio_j.htm

● SDL について

SDL(Simple Directmedia Layer)は，フリーなクロスプラットフォームの マルチメディア開発用 API である．SDL は，主にデバッグ用の画面を描画するために用いた．以下 SDL について解説する．

➢ 主な機能

SDL の主な機能は，グラフィックスの描画，キーボード入力，マウス入力，ユーザからの要求による終了といったイベント処理，オーディオの再生，CD-ROM オーディオの制御，タイマー等がある．今回使用したのは，主にグラフィックスを描画する機能である．

➢ 虫捕りゲームにおいて SDL を利用した箇所

虫捕りゲームでは，スクリーンの表示や，スクリーンの状態の更新，デバッグ用に文字を描画する必要のある箇所，キーイベントの部分で，SDL を使用した．

スクリーンのサイズ等についての情報やスクリーンの状態については，SDL_Surface 構造体が管理する．Game.cpp のコンストラクタで SDL_SetVideoMode 関数を呼び，スクリーンの幅，高さ，ピクセル深度等を設定し，SDL_Surface を作成している．State クラスでは，ゲーム中のスクリーンの表示を更新するために，SDL_Flip 関数を呼び，また，ClearScreen 関数内で一度スクリーン全体を黒く塗りつぶして，画面を再描画している．

文字の描画については，TTF_RenderText_Blended 関数でフォントや色の情報を保持したテキストを作成し，SDL_BlitSurface 関数で画面上に文字列を描画している． SDL ライブラリを利用せずに文字列を描画する場合は，WindowsAPI を利用して複雑で手間のかかる処理を記述しなければならないが，SDL ライブラリ使用することで，非常に少ない手間で文字列の描画をすることができる．

キーイベントについては，当初 SDLKey 構造体を使ってキーイベントの処理を行っていたが，SDLKey 構造体では，上下左右の方向キーが使用できなかったため，この構造体は使わないことにした．最終的には，SDL_GetKeyState 関数でキーの状態を監視し，指定されたキーが押されていたら，イベントを発生させるようにしている．また，イベントが発生する度にイベントキューに追加されていくので，State クラスでは, 定期的に SDL_PollEvent 関数を呼び, キューからイベントを取り出し，処理するようにしている．

➢ CriAudio との競合

SDL を導入する際に，一つ問題が発生した．Uint32 と Sint32 という名前の構造体が，CriAudio と SDL の双方で定義されていたため，定義が重複してしまい，コンパイルエラーになってしまった．この問題を回避するために，当初は片方の定義をコメントアウトしていたが，両方とも定義されている内容は同様だったために，すでに定義済みの場合のみ定義を有効にするように，プリプロセッサを記述した．

具体的には，以下のような記述をした．

```
#if !defined(_TYPEDEF_Uint32)
#define _TYPEDEF_Uint32
typedef unsigned long Uint32;
#endif

#if !defined(_TYPEDEF_Sint32)
#define _TYPEDEF_Sint32
typedef signed long Sint32;
#endif
```

このように記述することで，まだ構造体が定義されていないときのみ，定義を行うようにした．

➢ 参考 URL

下記の URL は，SDL のオフィシャルサイトの URL である．SDL のオフィシャルサイトから，SDL のダウンロードと，SDL についての情報を得ることが可能である．

http://www.libsdl.org/

● Boost について

Boost は，国際規格で定められた C++標準ライブラリの他に，さらに有用で移植性のあるライブラリを提供することを目的に，標準化委員会のメンバーによって作成された．以下 Boost について解説する．

➢ 主な機能

Boost は非常に多機能であり，様々なことができるが，代表的な機能としては以下のようなものが挙げられる．

✧ 自動的にオブジェクトを delete してくれるスマートポインタ

- ✧ 文字列・数値間のキャストや正規表現
- ✧ イテレータやコンテナ等のデータ構造とアルゴリズム
- ✧ グラフ，有理数，乱数，GCD，LCM，四元数等の数学関係の処理
- ✧ 日時・時刻・時間・時間経過等の制御

これらの機能うち，虫捕りゲームでは，文字列・数値間のキャストの機能を主に使用した．

➢ 虫捕りゲームにおいて Boost を利用した箇所

虫捕りゲームでは，主に文字列を数値に変換する，または数値を文字列に変換する必要のある箇所で，Boost の lexical_cast を用いた．虫捕りゲームでは，デバッグ用に SDL を利用して虫の座標や経過した時間等を描画するようにしているが，描画をするデータは文字列型として扱わなくてはならないため，数値を描画したい場合でもそれを一度整数型に変換する必要があった. しかし, C++の通常のキャストでは, 数値から文字列型への変換はサポートされていないので，Boost の lexical_cast を用いる必要があった. Boost の lexical_cast を用いることで, 文字列を数値に変換する, または数値を文字列に変換することができる.

➢ 参考 URL

下記の URL は，Boost のオフィシャルサイトの URL である．Boost のオフィシャルサイトから，Boost のダウンロードと，Boost についての情報を得ることが可能である.

http://boost.org/

4.1.6. **最終的なクラス構成とソースコードの規模**

- クラス構成

β版のクラス構成は，最終的に図 3-20 のようになった.

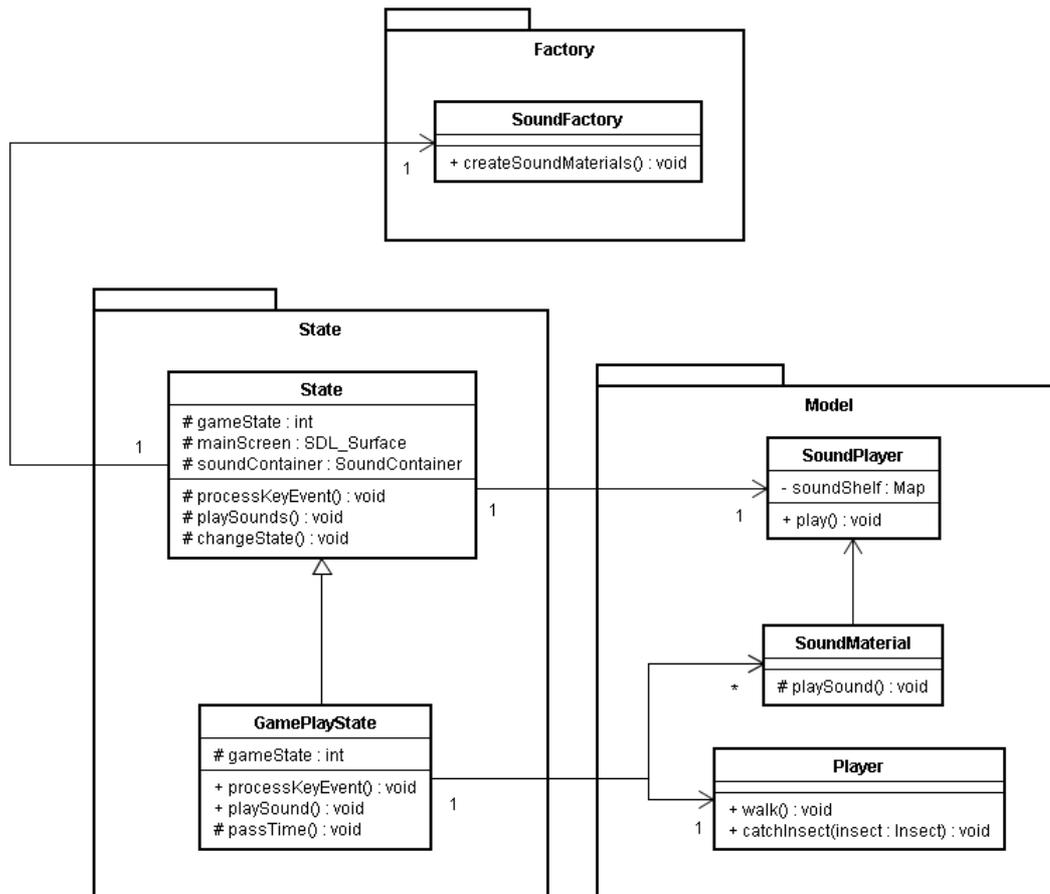



図 3-20：全体のクラス図

基本的な構造は, State クラスが SoundFactory クラスと SoundPlayer クラスを参照し, State クラスを継承した GamePlayState クラスが，SoundMaterial クラスと Player を参照する．SoundFactory クラス，State クラス，GamePlayState クラス，SoundMaterial クラスは，それぞれサブクラスを持つが，図 3-20 では省略している．

以下に，各パッケージのサブクラスを含めたクラス図を示す．

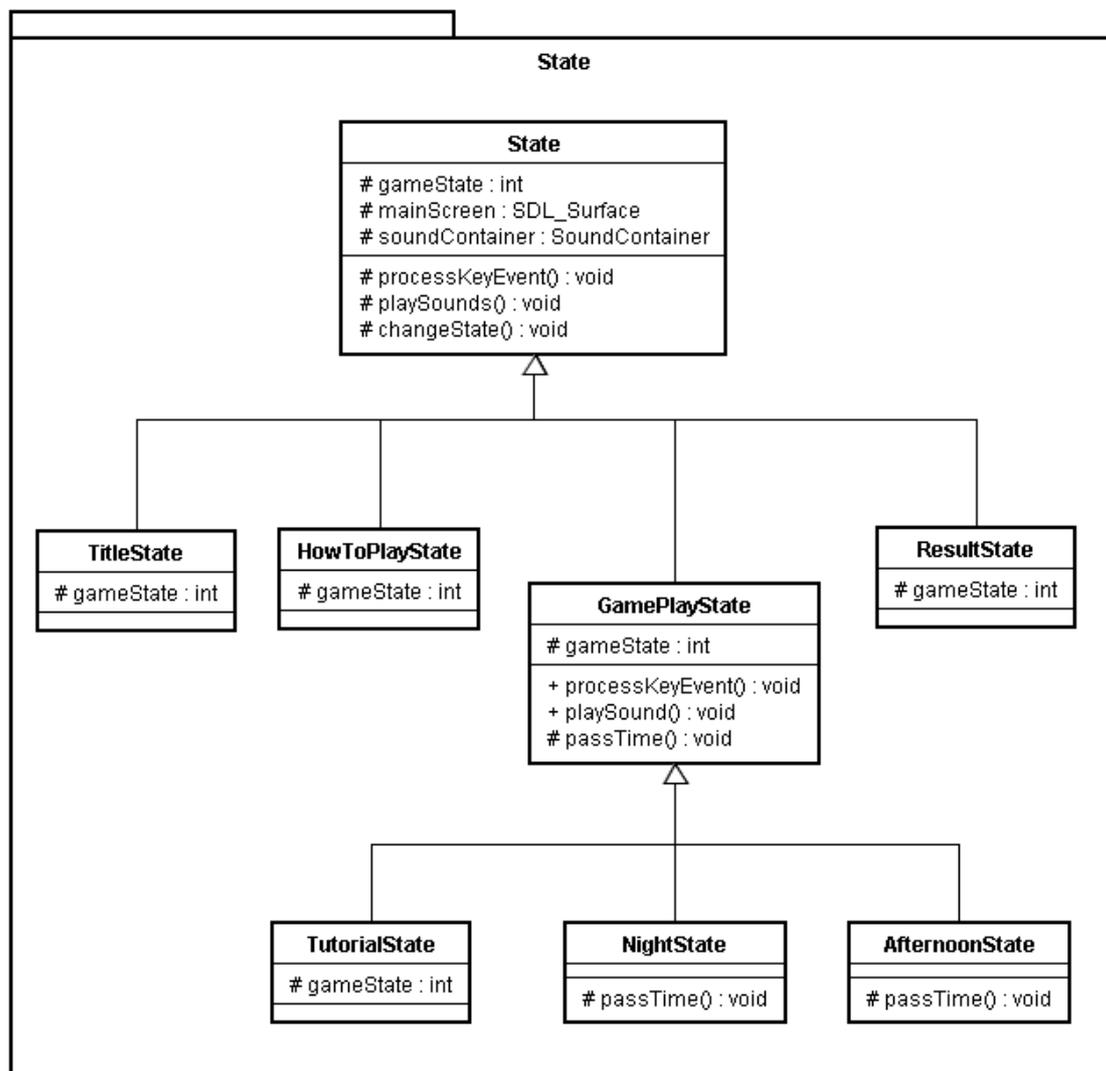



図 3-21：State パッケージ内のクラス図

図 3-21 は，State パッケージ内のクラス図である．各 State クラスは，ゲーム中のそれぞれの状態を管理する．各 State の遷移については，図 3-9 を参照していただきたい．

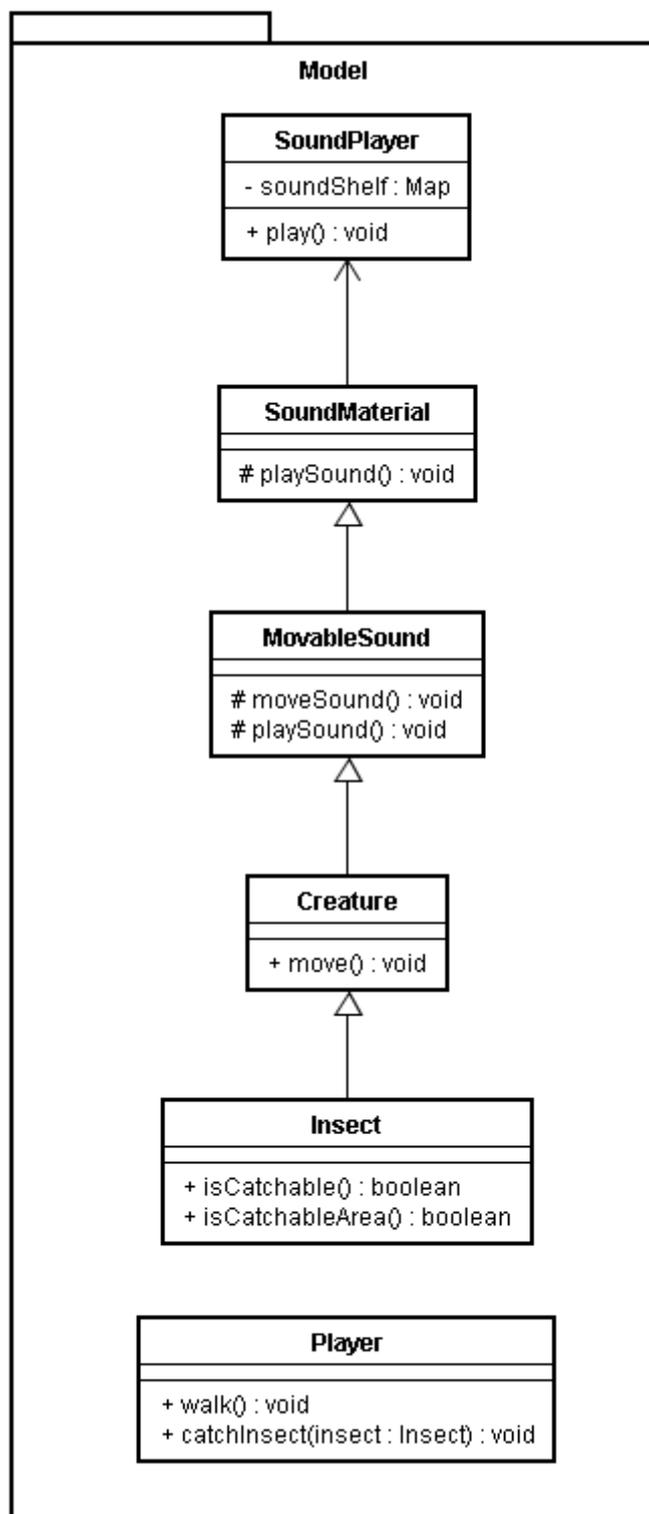



図 3-22：Model パッケージ内のクラス図

図 3-22 は，Model パッケージ内のクラス図である．SoundMaterial クラスが個々の音の

振る舞いを管理し，音を再生する・停止するといった処理は SoundPlayer クラスに委譲している．必要となる音の振る舞いによって，SoundMaterial クラスをベースに，継承して新たなクラスを作成している．

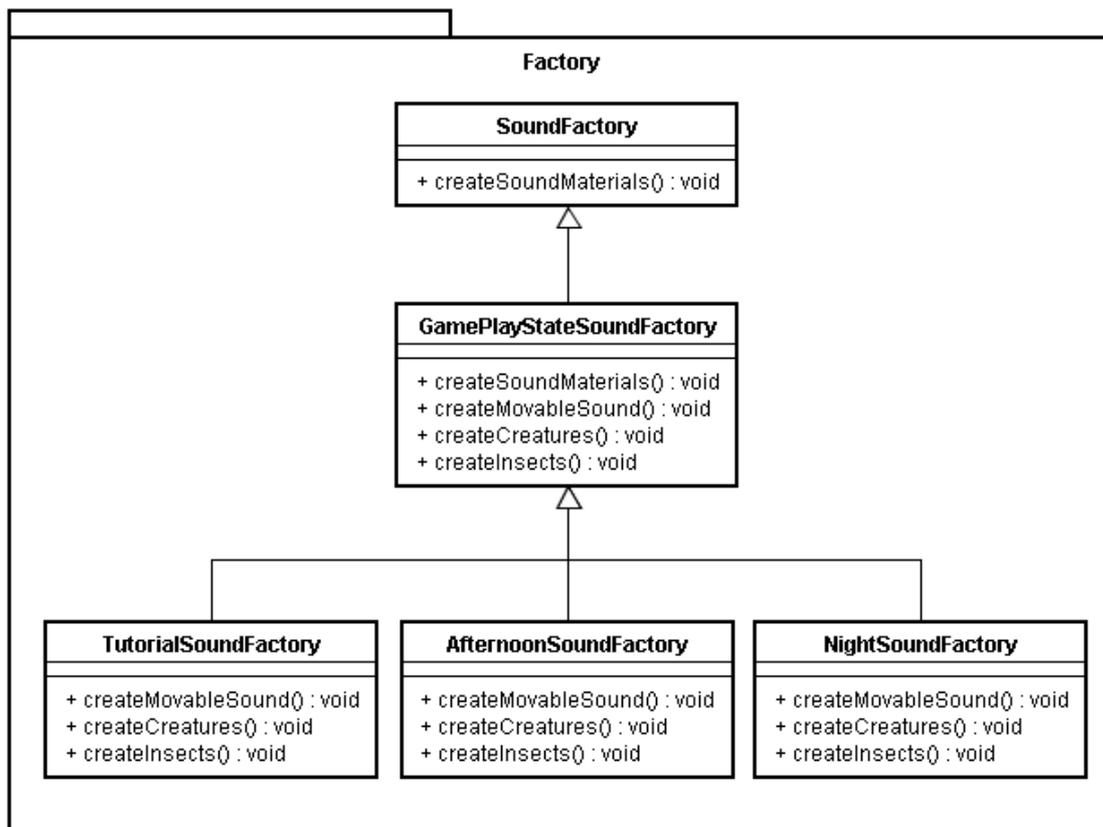



図 3-23：State パッケージ内のクラス図

図 3-23 は，Factory パッケージ内のクラス図である．State で使用する音の数が多く，生成が複雑になってしまうので，各 State クラスで必要な音の生成は，Factory クラスに委譲している．各 Factory クラスは，State パッケージのクラスごとに分割されている．TitleState クラス，HowToPlayState クラス，ResultState クラスは現在 Factory クラスを利用していないが，将来的に利用したくなったときのために，SoundFactory クラスを用意してある．なので，SoundFactory クラスを継承して各 State に必要な Factory クラスを作成することができる．

● ソースコードの規模

β版のソースコードの規模については，以下の通りである．

ファイル形式	行数	コメント	空行	Logic	ファイル名
cpp	75	20	9	46	State.cpp
cpp	26	18	3	5	TextCreator.cpp
cpp	77	16	10	51	TitleState.cpp
cpp	41	17	6	18	TutorialSoundFactory.cpp
cpp	199	38	32	129	TutorialState.cpp
cpp	205	56	28	121	Util.cpp
h	19	6	3	10	AfternoonSoundFactory.h
h	27	8	4	15	AfternoonState.h
h	32	6	9	17	Cage.h
h	37	11	6	20	Constants.h
h	37	8	7	22	Creature.h
h	31	6	6	19	Game.h
h	67	9	11	47	GamePlayState.h
h	27	7	4	16	GamePlayStateSoundFactory.h
h	22	6	3	13	GameStates.h
h	47	9	11	27	HowToPlayState.h
h	53	9	11	33	Insect.h
h	18	0	3	15	KeyFlag.h
h	33	0	7	26	KeyState.h
h	63	10	11	42	MovableSound.h
h	25	6	5	14	NightSoundFactory.h
h	27	8	4	15	NightState.h
h	64	4	14	46	OnMapObject.h
h	64	8	8	48	Player.h
h	63	11	14	38	ResultState.h
h	12	0	3	9	River.h

h	30	1	4	25	SDLCommonFunctions.h
h	36	0	7	29	Sound.h
h	38	5	8	25	SoundContainer.h
h	25	7	5	13	SoundFactory.h
h	39	6	6	27	SoundMaterial.h
h	44	5	7	32	SoundPlayer.h
h	52	6	7	39	State.h
h	62	5	8	49	StateMessages.h
h	24	10	4	10	TextCreator.h
h	26	7	5	14	TitleState.h
h	13	0	3	10	TutorialSoundFactory.h
h	40	7	7	26	TutorialState.h
h	38	6	9	23	Util.h
cpp	43	9	6	28	AfternoonSoundFactory.cpp
cpp	113	31	16	66	AfternoonState.cpp
cpp	45	17	6	22	Cage.cpp
cpp	63	20	7	36	Creature.cpp
cpp	54	10	9	35	Game.cpp
cpp	427	86	50	291	GamePlayState.cpp
cpp	53	19	7	27	GamePlayStateSoundFactory.cpp
cpp	194	34	26	134	HowToPlayState.cpp
cpp	1	0	1		InputDevice.cpp
cpp	138	30	16	92	Insect.cpp
cpp	44	15	4	25	KeyFlag.cpp
cpp	57	21	9	27	KeyState.cpp
cpp	34	5	6	23	Main.cpp
cpp	200	48	35	117	MovableSound.cpp
cpp	52	9	8	35	NightSoundFactory.cpp
cpp	150	33	18	99	NightState.cpp
cpp	85	24	13	48	OnMapObject.cpp

cpp	253	54	37	162	Player.cpp
cpp	318	70	36	212	ResultState.cpp
cpp	31	16	4	11	River.cpp
cpp	26	8	4	14	Sound.cpp
cpp	93	24	15	54	SoundContainer.cpp
cpp	13	6	2	5	SoundFactory.cpp
cpp	86	23	12	51	SoundMaterial.cpp
cpp	163	25	23	115	SoundPlayer.cpp

ファイル形式	行数	コメント	空行	Logic	ファイル数
cpp	3,359	802	458	2,099	31
h	1,235	197	224	814	33
合計	4,594	999	682	2,913	64

全体の規模については，プロジェクトメンバー各々が C++で開発したα版と比較して，3 倍以上大きくなっている．

また，β版の開発では設計書を作成しない代わりに，必要事項はソースコードにコメントとして記述し，ソースコードの可読性を高めるよう努めた．そのため，コメントの行数に関しては，C++で開発したα版と比較して約 5 倍多くなっている．また，ソースコード全体に占めるコメントの割合は，C++で開発したα版が約 15%だったのに対し，β版では約 22%に増加している．

4.2. β版評価

虫捕りゲームのβ版を，大岩研究室の方々，横浜市立盲学校の方々，株式会社ユードーの方々にプレイしていただき，アンケートに答えていただいた．アンケートの質問項目と選択肢は以下の通りである．

- 質問 1.「映像を用いないゲーム」をプレイした感想は？
  ① 非常に面白い
  ② 面白い
  ③ 普通
  ④ つまらない
  ⑤ 非常につまらない　の 5 段階

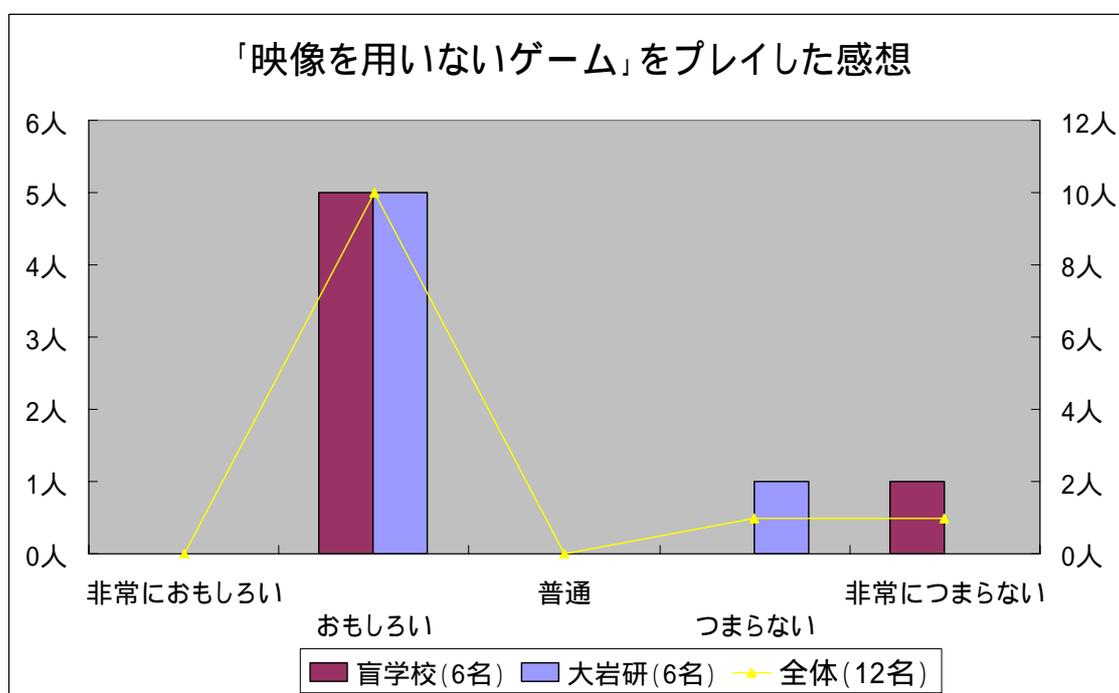
- 質問 2.「映像を用いないゲーム」をまたプレイしたいと思うか？
  ① 是非プレイしたい
  ② 機会があればプレイしたい
  ③ どちらでもよい
  ④ あまりプレイしたくない
  ⑤ 二度とプレイしたくない　の 5 段階

- 質問 3.「映像を用いないゲーム」をプレイして面白かった点は？（複数回答可）
  ① 映像を用いないという新しいゲームを体験できる
  ② 音に臨場感がある
  ③ 捕った虫の数やスコアを競うことが出来る
  ④ ゲームの難易度・バランスがちょうど良い
  ⑤ その他　の 5 項目

- 質問 4.「映像を用いないゲーム」について改善・追加した方がよいと感じた点は？（複数回答可）
  ① 虫や自然の音の聞こえ方
  ② 音声による説明
  ③ ゲームの操作性
  ④ ゲームの難易度・バランス
  ⑤ 虫捕り以外の要素を追加した方がよい
  ⑥ その他　の 6 項目

以上の 4 項目と，自由記述欄である．

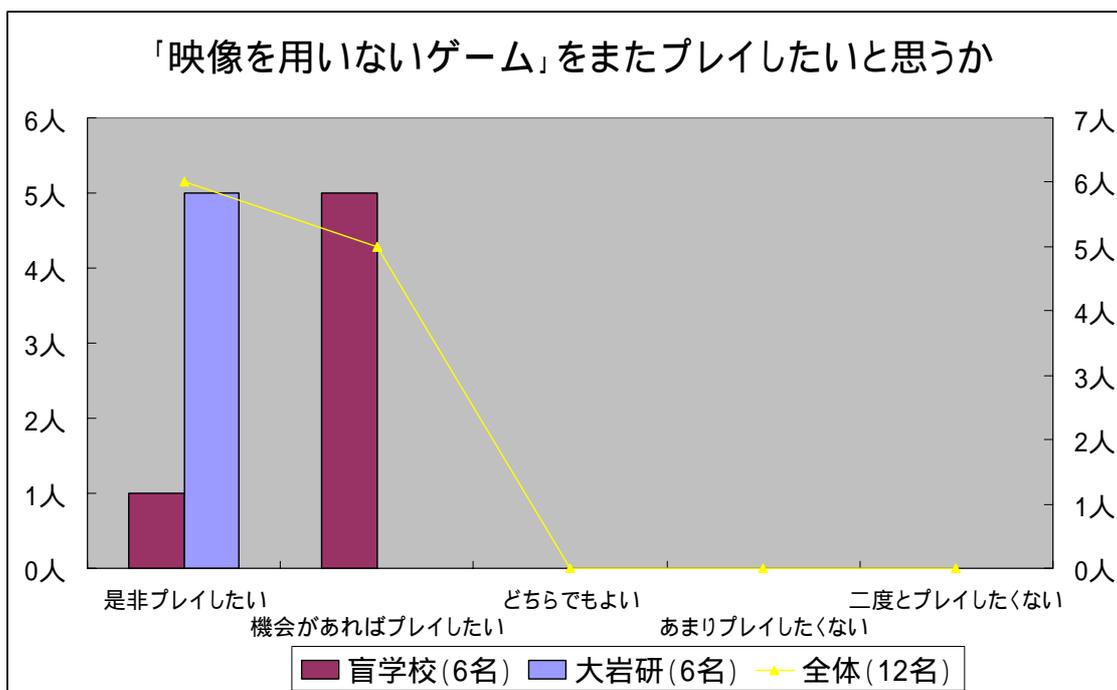
今期の「さうんど おんりぃ 2」プロジェクトでは，ゲームの開発が成功したか失敗したかを判断するための指標として，「ゲームを遊んでみて，面白い・もう一度プレイしてみたいと思う人の割合が，前期の『さうんど おんりぃ』プロジェクトよりも大きいかどうか」という判断基準を設けた．具体的には，前期の「さうんど おんりぃ」プロジェクトでは，面白いと思う人が 5 割，もう一度やりたいと思う人が 7 割という結果だったので，今期の「さうんど おんりぃ 2」プロジェクトでは，面白いと思う人が 7 割以上，もう一度やりたいと思う人が 8 割以上いるという数値を目標として設けた．以下，アンケートの結果を示す（添付資料：アンケート結果参照）．

表 3-3：「映像を用いないゲーム」をプレイした感想

「映像を用いないゲーム」をプレイしてみて，面白いと感じたという回答を行ったのが横浜市立盲学校の生徒・大岩研究室関係者ともに 6 人中 5 人で，その割合は 8 割 3 分であった．

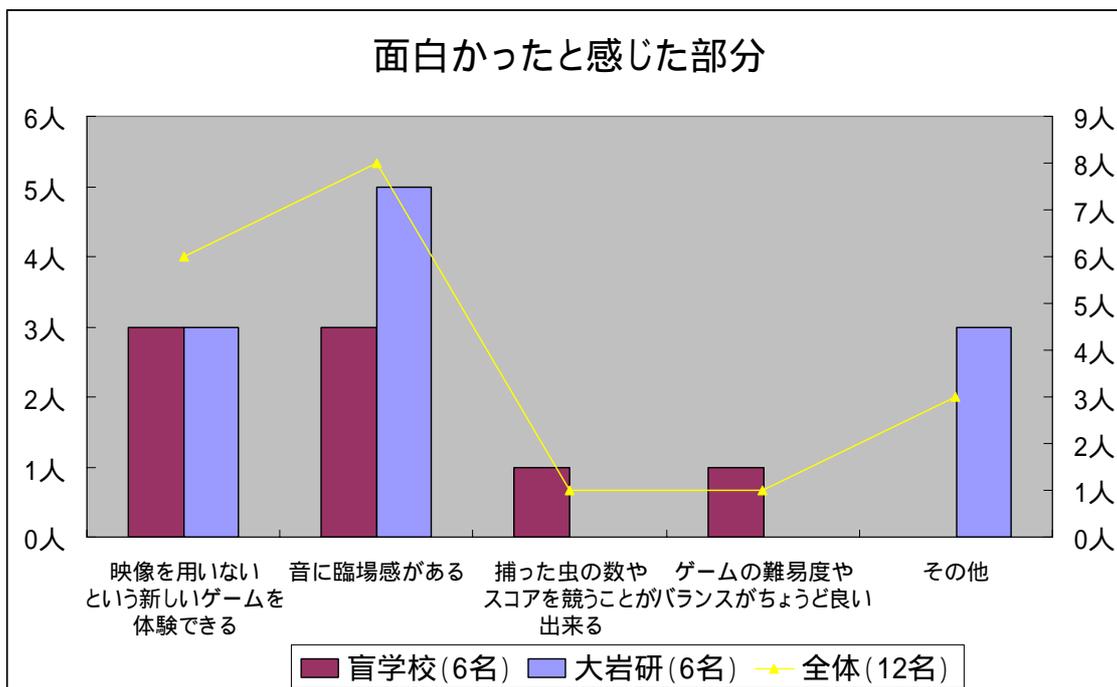
面白いと感じた人の割合は多かったが，非常に面白いという評価をした人はいなかった．これは，ゲームとしては未完成で十分に遊べるレベルではないが，今後開発が進んでゲームがより進化すれば，十分に遊べるものができそうであるという印象を受けた方が多かったということであると考えている．実際に，改良を加えれば非常に面白いものになりそうであるという意見は大岩研究室・盲学校の両方から多数頂いた．今後もより面白く遊んでもらえるゲームになるように改良を加えていきたい．

つまらない，非常につまらないと感じた人は，難易度が高くてなかなか虫が捕まらない

ことや，操作性の悪さにストレスを感じたようである．難易度が高い・操作性がよくないという指摘は盲学校・大岩研に関わらず多く寄せられたので，この点に関しては特に改善を要するところである．

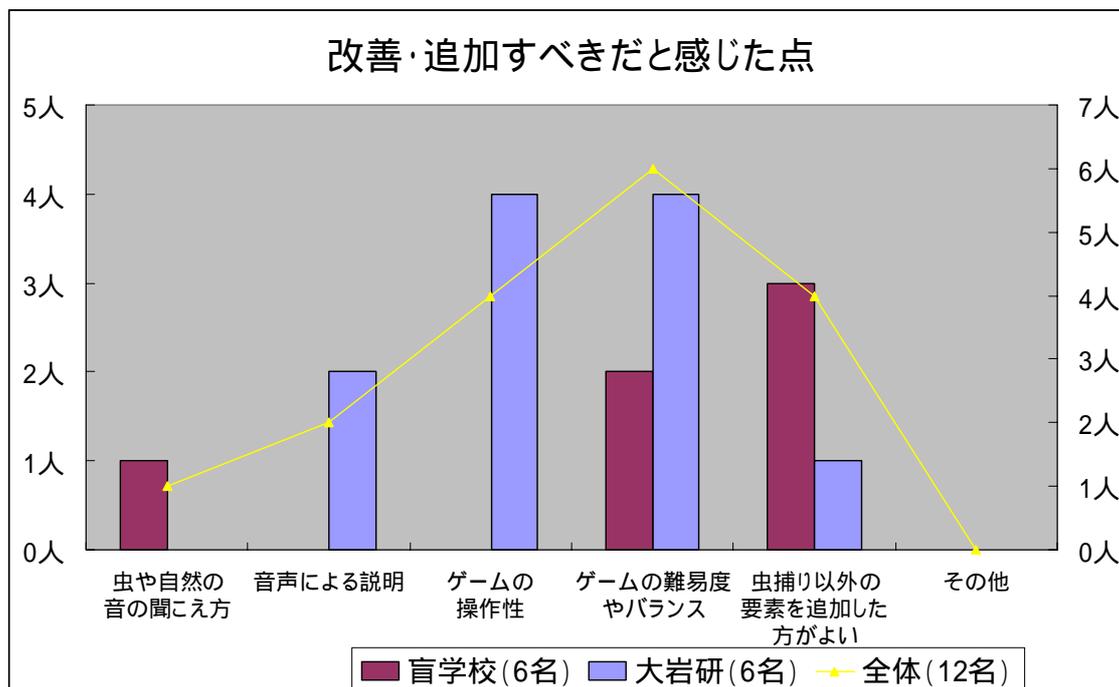
表 3-4：「映像を用いないゲーム」をまたプレイしたいと思うか

「映像を用いないゲーム」をプレイしてみて，是非またプレイしたい・機会があればプレイしたいと感じた人は，横浜市立盲学校の生徒・大岩研究室の関係者ともに 6 人中 6 人で，10 割であった．

全ての人が「是非プレイしたい」，もしくは「機会があればプレイしたい」という項目にチェックを入れた（どの項目にも記入していない人が一名いた．記入し忘れたものと思われる）．この結果から，今回作成した「映像を用いないゲーム」は，一度遊んだだけで飽きてしまうようなものではなく，何度も遊びたいと思えるようなゲームであるということができる．今後もより長く遊べるように改良を加え，繰り返し楽しんでもらえるようなゲームにしていきたい．

表 3-5：面白かったと感じた部分

「映像を用いないゲーム」をプレイしてみて，面白いと感じた点についてであるが，「映像を用いないという新しいゲームを体験できる」点が面白いと感じた人は，横浜市立盲学校の生徒・大岩研究室関係者ともに 3 人であった．「音に臨場感がある」点が面白いと感じた人は，横浜市立盲学校の生徒が 3 人，大岩研究室関係者が 5 人であった．「捕った虫の数やスコアを競うことができる」点が面白いと感じた人は，横浜市立盲学校の生徒が 1 人であった．「ゲームの難易度・バランスがちょうど良い」点が面白いと感じた人は，横浜市立盲学校の生徒が 1 人であった．「その他」の点が面白いと感じた人は，大岩研究室関係者が 3 人であった．

「音に臨場感がある」と感じた人は，横浜市立盲学校の生徒の方々よりも，大岩研究室の方々の方が多かった．横浜市立盲学校の方々は，日頃から聴覚を研ぎ澄ませて生活しているため，音に関する感覚が鋭く，少しでも不自然な点があるととても気になってしまうため，大岩研究室の方々よりも臨場感を感じられなかったのではないかと考えられる．

「捕った虫の数やスコアを競うことができる」と，「ゲームの難易度・バランスがちょうど良い」にチェックを入れた方は，それぞれ一人しかいなかった．そのため，これらの項目に関しては，より改善していく必要があると感じている．

表 3-6：改善・追加すべきだと感じた部分

「映像を用いないゲーム」をプレイしてみて，改善・追加すべきだと感じた部分についてであるが，「虫や自然の音の聞こえ方」を改善すべきだと感じた人が，横浜市立盲学校の生徒が 1 人であった．「音声による説明」を改善・追加すべきだと感じた人は，大岩研究室関係者が 2 人であった．「ゲームの操作性」を改善した方がよいと感じた人は，大岩研究室関係者が 4 人であった．「ゲームの難易度・バランス」を改善すべきだと感じた人は，横浜市立盲学校の生徒が 2 人，大岩研究室関係者が 4 人であった．「虫捕り以外の要素を追加した方がよい」と感じた人は，横浜市立盲学校の生徒が 3 人，大岩研究室関係者が 1 人であった．

「音声による説明」や，「ゲームの操作性」を改善・追加すべきであるという項目にチェックしたのは，大岩研究室関係者が多かった．大岩研究室では，使いやすいソフトウェアを開発することを心がけている方が多いので，ソフトウェアの使い方の説明や，ゲームの操作性に関しては，シビアな視点を持っている方が多い．そのため，これらの項目について不満を持った方が多かったのではないかと考えられる．しかし，シビアな目を持つ方々にも満足していただける品質のゲームを作成するために，今後さらなる改良を加えていきたいと考えている．

「ゲームの難易度・バランス」については，全体的に不満を持った方が多かった．「難しすぎる，どこまで虫に近づいたら捕まえられるのかよくわからない」といった意見が多数寄せられたので，「ゲームの難易度・バランス」については特に改善を要する箇所であると思われる．具体的な改善案として，誰でも簡単に虫を捕まえられる初心者モードを追加し，

慣れた人はより難しいモードを選択できるようにするということが上げられる．

アンケートの結果，面白いと思う人が 8 割 3 分，もう一度やりたいと思う人は 10 割であった．そのため，面白いと思う人が 7 割以上，もう一度やりたいと思う人が 8 割以上いることという当初の目標は完全に達成された．

4.2.1. **横浜市立盲学校の生徒の方々からの意見・指摘**

横浜市立盲学校の生徒の方々からいただいた主な意見・指摘は，以下のようなものである．それぞれの意見・指摘について，横浜市立盲学校図書館司書教諭の松田基章様から補足意見をいただいた．

● 虫以外の音が目立ちすぎていて，虫の鳴き声がよくわからなくなってしまう

この指摘について，松田教諭から，「視覚にハンディキャップを有する方々は，視覚から情報を得ることができない代わりに，聴覚に頼って情報を得ていることが多い．そのため，聴覚が一般の人に比べて発達している人が多く，少しでも音に違和感があったり，煩雑だったりすると，非常に不快である．そのため，ゲーム中の音のバランスの悪さが気になってしまったのではないか.」という意見をいただいた．この点については，自分たちがプレイしたときも，注意深く聞いてみると，音によって音量の大きさにバラつきがあったり，音に不自然さがあったりすることがわかる．何気なくプレイしていると中々気付きづらい点であるが，このゲームの善し悪しを決める非常に大きな要素なので，少しでも不自然に感じる点は改良していく必要がある．

● ゲームの難易度が高すぎる

この指摘について，松田教諭から，「普段あまりゲームに触れる機会のない生徒も居るので，初めは非常に簡単なところから始めて，慣れてきたら徐々に難易度を上げていけるようにした方が楽しめるのではないか.」という意見をいただいた．この点に関しては，視覚にハンディキャップを持つ方に限らず，多くの方から同じような指摘をいただいているため，大いに改善を要する点である．このゲームでは，簡単な練習ができるようにはなっているが，ゲーム自体の難易度の設定等ができず，慣れないと中々虫が捕まえられず，ストレスやフラストレーションがたまってしまう．そのため，最初は虫が逃げなかったり，移動範囲が狭かったり，虫を捕るタイミングをアドバイスしてくれる等の簡単なモードを作成した方がよかった．

● 弱視の方が対象であっても，映像があった方がよい

この指摘について，松田教諭から，「横浜市立盲学校の中では，全盲の方の割合は低く，

弱視の方の割合の方が高い．そのため，弱視の方にとっては，ゲームに映像が付いていた方が面白いと感じるのだろう．また，視覚にハンディキャップを有する方々は，視覚にハンディキャップを有する方のみを対象に作られたゲーム，あるいは健常者が遊ぶ気にならないようなゲームには，魅力を感じない傾向がある．多くの人にとって魅力的なものを，視覚にハンディキャップを持つ方々も好む傾向がある．」という意見をいただいた．本プロジェクトのコンセプトとして，「音だけを使って健常者も視覚障害者も同じように楽しめるゲーム」を目指して開発していたのだが，映像がないとどうしても印象が地味になってしまい，魅力を感じにくくなってしまう面がある．この点に関しては，「ゲームには音だけしか使用しない」という方針を変更して多少の映像を使用するか，あるいは音だけを使ってより魅力的なゲームになるように改良していく必要がある．

4.2.2. **大岩研関係者の方々からの意見・指摘**

大岩研関係者の方々からいただいた主な意見・指摘は，以下のようなものである．

- 距離感がわかりづらく，自分がどこにいるのかよくわからない

この点に関しては，移動をしているうちに自分がどちらの方角を向いているのか，マップ上のどの辺にいるのかがわからなくなってしまい，迷ってしまうことが多いというものである．現状のゲームでは，プレーヤーの方向や位置を知らせる仕組みがないため，混乱を招いてしまっているのだと考えられる．そのため，一定の方向から目印となるような音を鳴らしたり，虫に対しても捕まえられる範囲にいたら知らせてくれたりするような仕組みを用意して改善したいと考えている

- ゲームの難易度が高すぎる

この点に関しては，横浜市立盲学校の生徒の方々からいただいた指摘と共通である．現状のゲームは，誰がやっても難しいと感じてしまうようなので，改善したい．

- 移動できる範囲が狭い

この点に関しては，行き当たりばったりで移動しているとすぐに行き止まりになってしまい，戻らなくてはならないので，鬱陶しいと感じる方が多かったようである．この点に関しては，Config.txt でマップの広さを指定できるようになっているので，Config.txt の値をユーザが書き換えることでマップの広さを変更することができる．大岩研究室の関係者の方々にプレイしていただいた際には，その点をアナウンスし忘れてしまったので，アナウンスすべきであったと反省している．

● もっと虫のバリエーションが欲しい
この点に関しては，まだ捕まえられる虫が少ないので，捕まえられる虫の種類を増やしたいと思う．また，マップ上のエリアによって生息している虫の種類が異なっている等の工夫があるとより面白くなるのではないかという意見もいただいたので，参考にさせていただきたい．

● 虫を捕まえたことに対する感動があまり感じられない
この点に関しては，虫を捕っても「○○を捕まえた」という声だけで，もっと派手な演出が欲しいというものである．例えば，虫を捕まえたら，捕まえた虫に関する豆知識等を教えてくれたり，捕まえた虫の数によってスコアだけでなく，ランクも付けて，より高いランクを目指せるようにしたりすると，虫を捕まえるモチベーションが上がるのではないかという意見をいただいた．

4.2.3. クライアントからの意見・指摘

本プロジェクトのクライアントである株式会社ユードーの南雲玲生代表取締役から頂いた主な意見・指摘は，以下の通りである．

● 今回のテーマであった，演出や企画の質の向上
この点について，南雲代表取締役から，「具体的に，屋久島の音や，虫の効果音などの使用により，空間の演出がうまくできています．FPS の設計も理解できます．これをベースに，都市などのバージョンができますね．音が変わるだけでも，ユーザへ想像性や，美しさ，心地よさを提供できます．」というコメントをいただいた．空間の演出についてはうまくいっているという評価をいただけたので，ゲームの幅を広げていくために，今後はより様々な場面を再現することを考慮していきたい．

● ゲームバランスの調整が必要
この点について，南雲代表取締役から，「しばらく慣れるまでに，物標とプレーヤーの距離感がつかめません．どの音量レベルだと，目の前にいるという感覚が難しいのです．感覚をユーザへ与えるには，ゲームの遊び方，や，ゲーム開始時に，目の前に人を登場させ，この人が，”ゲーム開始”というアナウンスや，プレーヤーの属性をアナウンスします．その人との音量の距離感をデフォルト値にして，ゲームプレイヤーは，進められるのでしょう．」というコメントをいただいた．南雲代表取締役も，ゲームのバランスや操作性に問題を感じたようなので，この点については最優先で改善を要することを再認識させられた．また，「目の前に人を登場させてアナウンスさせることで，ゲームをプレイする人に距離感を教える」というアイデアを提案していただいたので，ゲーム改善の際には是非参考にさせて

いただきたい.

● サウンドの制作環境の向上

この点について，南雲代表取締役から，「基本的に，音の所在は，音のアタック（立ち上がりが早い）が強い，音圧が高いとわかり易いはずです．一部，存在が弱い音の虫がいた用に思います．あと，サウンドの編集ですね．品質の高い機器で，編集を行うと，圧倒的にクオリティが変わります.」というコメントをいただいた．β版のゲームで使用している音は，フリーで利用できるツールを使用して編集したものである．そのため，高度な編集ができず，音のクオリティは決して高いものではなかった．南雲代表取締役からは，株式会社ユードーの機材をお貸ししてくださると申し出ていただけたので，南雲代表取締役と相談しながら，よりクオリティの高い音を作成していきたい.

● ゲームイメージ（ブランド）の確立

この点について，南雲代表取締役から，「ゲームプランニングになりますが，ユーザに，想像力を提供できるゲーム名や，ストーリーを提供してもよいのでは．森ではなく，せっかく屋久島の音ですので，屋久島の冒険とか．インパクトが与えられますね.」というコメントをいただいた．β版のゲームでは, 名もない森の中で漠然と虫を捕まえるだけなので, もっと具体的な場所で，なおかつ多くの人が魅力を感じるような場所を音で再現できるとよりおもしろくなりそうである．このようにゲームのイメージを鮮明にすることは重要なポイントであると思うので，今後考えていきたい.

4.2.4. 「虫捕りゲーム」から得られたもの

● ゲーム開発に向いているプロセス

今回の「さうんど おんりぃ２」の開発を行うにあたり，ゲーム開発特有のプロセスを用いた．企画書・仕様書は面白いと思う要素が思い浮かべば，どんどん改変していった．また，設計書は最低限手書きのメモ程度に残し，実装の時間をできるだけ多く捕ることでゲームを開発しながらその面白さを追及した.

前回のプロジェクトでは，PMBOK に従って要件定義，設計，実装，評価というフェーズを通じて，ソフトウェア開発手法を用いて「映像のないゲーム」を開発した．その結果，プロジェクトとしては期間内にスコープで定義したものが出来上がり成功を収めたが，ゲームとしてはそれほど面白いものが出来上がらなかった.

今回, ゲーム特有の開発手法を実践したことが, そこにはソフトウェア開発にはない様々な発見があった．詳細については個人レポートを参照して頂きたいが，ソフトウェア開発手法のスケジュール管理やタスク管理方法と，ゲーム開発手法の設計書に振り回されない実装方法をうまく組み合わせることで，ゲーム開発により適した新しいプロセスを組み合

わせて主観的にも客観的にも面白いと評価することが出来るゲームを開発できたことが大きな成果である．

● 「映像のないゲーム」のアイデアを実現するためのツール・環境の開発

虫捕りゲームは音を変えるだけで，比較的簡単に色々なゲームを開発することができる．たとえば，海中の音，砂漠の音などを使えば，様々な自然環境で魚や虫を捕まえるというゲームに出来る．また，発想を変えて，森の音を商店街の雑踏の音に変え，動物や虫の声を店先で商品を売る店員の声にすれば，店を探して商店街を歩き回るといったゲームも作ることができる．

つまり，虫捕りゲームの存在によって「映像のないゲーム」に様々な可能性を見出すことができるのである．虫捕りゲームはゲームとしてはまだまだ改良の余地が多く残されているが，「映像のないゲーム」の基礎が出来たことで，今回他にも挙がった様々な企画をゲームとして実現する目処が立った．色々な人が思い浮かんだ「映像のないゲーム」のアイデアを実現させることができるツールや環境の開発ができたことが，「虫捕りゲーム」開発の大きな成果である．

第5章. 添付資料

- 全体スケジュール
- 見積もりと実績
- ミーティングログ
- 進捗報告会資料
- 週報
- ソースコード（音記憶ゲーム：Java）
- ソースコード（聖徳太子ゲーム：Java）
- ソースコード（さうんどシュート：C++）
- ソースコード（聖徳太子ゲーム改：C++）
- ソースコード（ForestWalking：C++）
- アンケート結果
- プロジェクト宣伝用資料